国家精品课程教材
国家级精品资源共享课教材
普通高等教育“十三五”规划教材
研究型教学模式系列教材

C 语言程序设计

(第 3 版)

蒋　彦　韩玫瑰　主编

张芊茜　黄艺美　崔忠玲　李崇威　编

刘明军　主审

電子工業出版社
Publishing House of Electronics Industry

北京·BEIJING

内 容 简 介

本书是国家精品课程教材、国家级精品资源共享课教材，以任务驱动的方式，通过实例讲授程序设计的基本概念和基本方法，把重点放在解题思路上，试图贯穿以程序编写带动语法教学的模式，引导读者掌握 C 语言的核心编程方法，提高应用能力。本书共 7 章，主要内容包括：C 语言程序基础、程序基本结构、模块化程序设计、简单构造数据类型、复杂构造数据类型、磁盘数据存储、实用程序设计技巧等。本书配套《C 语言程序设计实验教程（第 3 版）》，并提供课程网站、电子课件、习题答案及程序源代码。

本书可作为高等学校本科生教材，也可作为专科和高职高专教材及计算机等级考试的参考书，还可供相关领域的工程技术人员学习参考。

图书在版编目（CIP）数据

C 语言程序设计/蒋彦，韩玫瑰主编. —3 版. —北京：电子工业出版社，2018.3
ISBN 978-7-121-33770-3

Ⅰ. ①C… Ⅱ. ①蒋… ②韩… Ⅲ. ①C 语言－程序设计－高等学校－教材 Ⅳ. ①TP312.8

中国版本图书馆 CIP 数据核字（2018）第 037875 号

策划编辑：王羽佳
责任编辑：王羽佳
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×1 092　1/16　印张：14　字数：455 千字
版　　次：2007 年 1 月第 1 版
　　2018 年 3 月第 3 版
印　　次：2018 年 3 月第 1 次印刷
定　　价：39.90 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888，88258888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

本书咨询联系方式：（010）88254535　wyj@phei.com.cn。

第 3 版前言

济南大学开设“C 语言程序设计”课程已有 20 多年的历史，在课程组全体老师的共同努力下，该课程 2005 年被评为山东省精品课程，2009 年被评为国家精品课程，2016 年被评为国家级精品资源共享课。

2007 年我们编写了本课程教材《C 语言程序设计》，2011 年修订了《C 语言程序设计（第 2 版）》。教材出版以后，被多所高等学校选作教材，并被数十所高校选作教学的主要参考书。近年来，陆续收到了各位同仁和广大读者给予的高度评价，以及一些很好的修订建议。七年后，我们根据在教学过程中的实际感受，结合收集到的建议和意见，对第 2 版教材进行了修订，出版了《C 语言程序设计（第 3 版）》和《C 语言程序设计实验教程（第 3 版）》。

第 3 版的主要修订内容体现在以下几个方面：

① 作为《大学计算机》课程的衔接，用 C 语言实现《大学计算机》课程中的算法与程序设计思想，有利于加强学生计算思维能力的锻炼与培养。

② 结合 ACM 竞赛的参赛经验，把 ACM 竞赛模式引入到教学中，建立了适合本课程教学的 OJ（Online Judge，在线判题）系统，把课程中的全部例题、习题、实验改编为 ACM 模式，提供在线练习，系统实时返回运行结果。

③ 指针作为 C 语言的特色，也是该课程的难点之一，大多教材都作为独立的一章进行讲解。我们认为指针也是变量中的一种，只是有其特殊性。因此，将指针的概念和有关内容分布到教材的多个章节中进行讲解，淡化了指针的独立性。希望这样更有利于难点的分解，有助于读者逐步建立起指针的概念，掌握指针的使用方法。

④ 从模块化程序设计的思想引入函数，调整了全书结构，使结构层次更清晰，并以实例的方式说明模块化程序设计的优点。对于文件操作，增加了实用例题，使其更具有指导性。实用程序设计技巧更侧重于实用性和技巧性，并针对 Visual C++ 6.0 环境设计了例题。

⑤ 将程序移植到 Visual C++ 6.0 环境中调试、运行。所有例题均在 Visual C++ 6.0 下调试通过。

本书配套《C 语言程序设计实验教程（第 3 版）》，并提供电子课件、习题参考答案及程序源代码，请登录华信教育资源网 http://www.hxedu.com.cn 注册下载。

本书由蒋彦、韩玫瑰统稿，其中第 1～2 章由蒋彦、刘明军修订，第 3 章由张芊茜、李崇威修订，第 4～5 章由崔忠玲、黄艺美修订，第 6～7 章由韩玫瑰修订，全部习题由蒋彦修订。全书由刘明军教授审定。

在本书的修订过程中，得到了全体课程组老师的关心与支持。本书的编写参考了大量近年来出版的相关书籍及技术资料，吸取了许多专家和同仁的宝贵经验。在此一并表示衷心的感谢！

尽管我们做出了很大的努力，修订了第 2 版的诸多不足之处，但由于水平有限，书中难免还有错误或不妥之处。我们恳请各位专家、同行及广大读者批评指正！

作　者

2018 年 2 月

第 2 版前言

济南大学开设“C 语言程序设计”课程已有十余年的历史，在课程组全体成员的共同努力下，该课程 2005 年被评为山东省精品课程，2009 年被评为国家精品课程。

2007 年我们编写了本课程教材《C 语言程序设计》，在电子工业出版社出版。该教材出版以后，被多所高等学校选作教材，并被数十所高校作为教学的主要参考书使用。在使用过程中，各位同仁和广大读者给予了较高的评价，并提出了很好的建议。四年后，我们根据在教学过程中的实际感受，结合收集到的建议和意见，对第 1 版教材进行了修订，出版了《C 语言程序设计（第 2 版）》和《C 语言程序设计实验教程（第 2 版）》。

第 2 版的主要修订内容体现在以下几个方面：

① 指针作为 C 语言的特色，也是该课程的难点之一，一般的教材都作为独立的一章进行讲解。我们认为指针也是变量中的一种，只是有其特殊性。因此将指针的概念和有关内容分布到教材的多个章节中进行讲解，淡化了指针的独立性。希望这样可以有利于难点的分散，有助于读者逐步建立起指针的概念，掌握指针的使用方法。

② 从模块化程序设计的思想引入函数，调整了全书结构，使结构层次更清晰，并以实例的方式说明模块化程序设计的优点。

③ 对于构造数据类型，除更清楚地介绍基本概念和使用方法外，重点介绍与指针的关系，并结合实例，突出将其作为函数参数的使用方法，与指针内容紧密结合。

④ 对于文件操作，增加了实用例题，使其更具有指导性。实用程序设计技巧更侧重于实用性和技巧性，并针对 Visual C++ 6.0 环境设计了例题。

⑤ 将程序移植到 Visual C++ 6.0 环境中运行、调试。所有例题均在 Visual C++ 6.0 下调试通过。

本书配套《C 语言程序设计实验教程（第 2 版）》，并提供电子课件、习题参考答案及程序源代码，请登录华信教育资源网 http://www.hxedu.com.cn 注册下载，或登录国家精品课程网站 http://c.ujn.edu.cn/。

本书由刘明军统稿，其中第 1～2 章由刘明军修订，第 3 章由李崇威修订，第 4～5 章由黄艺美修订，第 6～7 章由韩玫瑰修订，部分章节及全部习题由蒋彦修订。

在本书的修订过程中，得到了全体课程组教师的关心与支持。本书的编写参考了大量近年来出版的相关书籍及技术资料，吸取了许多专家和同仁的宝贵经验。在此一并表示衷心的感谢！

尽管我们做出了很大的努力，修订了第 1 版的诸多不足之处，但由于水平有限，书中难免还有错误或不妥之处。我们恳请各位专家、同行及广大读者批评指正！

作　者

2011 年 1 月

前　言

C 语言是一种被广泛学习、普遍使用的计算机程序设计语言。它的高级语言形式、低级语言功能的特点具有特殊的魅力，因而被大多数高等学校采用作为典型的计算机语言教学课程，也被选为计算机等级考试、全国计算机应用证书考试等多种计算机技能考试的考试内容。此外，C 语言作为一门实用且功能强大的程序设计语言，被程序设计人员广泛使用。因此，C 语言是一门十分重要的程序设计语言。

学习程序设计语言的目的在于使用该语言编写程序解决实际工作中的问题，而不仅仅是掌握该语言的语法。读者在 C 语言程序设计的学习中，很容易陷入语法的泥潭中，一开始就学习大量的语法知识，难以记忆，更难以理解，难免产生为难情绪，感到 C 语言难学，更不知如何使用。作者在建设“省级精品课程”的同时，对课程的教学理念和教学方法进行了认真思考，集聚多年的教学经验，从而完成了本书——一本真正注重培养读者程序设计能力的教材。

本教材具有以下特色：

① 进行了基于任务驱动的实例教学法的尝试。以任务驱动的方式，通过实例讲授程序设计的基本概念、基本方法，把重点放在解题思路上。

② 从开始学习就使读者将注意力集中在所解决的问题领域，从具体实例理解 C 语言的开发特点和程序总体框架，通过实例本身学习某一类问题的解决方法和计算设计，贯穿以程序编写带动语法教学的模式。

③ 在 C 语言的环境下，针对实际问题进行分析，构建数学模型，设计算法，最后编程实现。

④ 将原来的被动填鸭式的灌输语言知识，变为自主的学习和探索，读者不再陷入语言的语法规则，而重在程序设计方法的学习和探究，通过编程掌握语言。

⑤ 在学习的不同阶段设计不同的针对性的实际，从而得到较好的教学效果。例如，开始阶段设计的实例是将学生的注意力吸引在 C 语言的总体功能和程序的总体框架上；在学习中间阶段设计针对某些数据类型或应用特点的实例、针对模块分解和组合的实例、针对算法分析与设计的实例等；在学习的后期进行综合课程设计，将所学知识融会贯通。

⑥ 引导读者掌握 C 语言的核心编程方法，提高应用能力。引导学生在解题编程的实践中探索其中的规律，将感性认识升华到理性高度。

全书分为理论部分和实验部分。理论部分共分 7 章。第 1 章介绍 C 语言的基本结构、语法成分、调试方法，简单 C 语言程序的设计、输入/输出语句等：第 2 章结合实际问题介绍 C 语言程序的基本结构；第 3 章介绍如何将复杂问题简单化处理的编程方法；第 4、5 章介绍如何编写具有构造数据类型的程序；第 6 章介绍磁盘数据存储程序的设计方法；第 7 章介绍实用程序设计的一般方法。

通过学习本书，你可以：

- 以任务驱动的方式了解 C 语言程序设计的基础知识

- 掌握 C 语言的核心编程方法
- 以程序编写带动语法的学习
- 建立程序设计的思想
- 通过上机实验提高程序设计能力
- 小试身手——利用 C 语言进行程序设计

本书的编写工作由刘明军主持。第 1、2 章由刘明军编写，第 3 章由王卫峰编写，第 4、5 章由黄艺美编写，第 6、7 章由韩玫瑰编写，实验部分由蒋彦和韩玫瑰老师编写，潘玉奇老师参加了部分内容及部分习题的编写。全书由刘明军统稿。中国石油大学（华东）的魏东平教授在百忙之中审阅了全书，济南大学的董吉文教授对本书的编写提出了宝贵意见。本书的编写参考了近年来出版的大量书籍及相关技术资料，吸取了许多同仁和专家的宝贵经验，在此一并表示衷心的感谢！

在琳琅满目的书海中，编写一本有特色并能使读者感兴趣的教材绝非易事。尽管我们付出了很大的努力，但由于水平有限，书中难免出现错误或不妥之处，我们诚恳地欢迎读者和同行批评指正。

作　者

2007 年 1 月

目　录

第 1 章　C 语言程序基础 ……（1）
1.1　C 语言程序的基本结构……（1）
1.1.1　认识 C 语言程序……（1）
1.1.2　基本结构 ……（3）
1.1.3　程序举例 ……（4）
1.2　C 语言基本语法成分……（6）
1.3　C 语言数据类型 ……（8）
1.3.1　基本数据类型 ……（9）
1.3.2　指针类型 ……（11）
1.3.3　构造数据类型 ……（12）
1.4　C 语言的表达式和语句……（12）
1.5　C 语言程序运行过程……（13）
1.5.1　源程序、目标程序和可执行程序的概念 ……（13）
1.5.2　C 语言程序的开发步骤 …（14）
1.6　编写简单的 C 语言程序……（15）
1.7　数据的输入与输出 ……（16）
1.7.1　printf 函数 ……（17）
1.7.2　scanf 函数……（18）
1.8　C 语言中的宏定义……（20）
1.8.1　不带参数的宏定义……（20）
1.8.2　带参数的宏定义……（21）
1.9　算法……（23）
1.9.1　算法的概念及特性……（23）
1.9.2　算法的表示方法……（24）
1.10　C 语言的产生、发展及特点 …（26）
1.10.1　C 语言的产生及发展 ……（26）
1.10.2　C 语言的特点……（27）
习题 ……（28）
第 2 章　程序基本结构……（33）
2.1　分支结构……（33）
2.1.1　单分支结构 ……（33）
2.1.2　双分支结构 ……（34）
2.1.3　多分支结构 ……（35）
2.1.4　if 语句的嵌套……（36）
2.1.5　条件运算符 ……（38）
2.1.6　switch 语句……（39）
2.2　关系运算和逻辑运算……（40）
2.2.1　关系运算符和关系表达式 ……（41）
2.2.2　逻辑运算符和逻辑表达式 ……（41）
2.3　循环结构……（42）
2.3.1　概述 ……（42）
2.3.2　当型循环 while……（43）
2.3.3　直到型循环 do-while……（45）
2.3.4　当型循环 for……（46）
2.3.5　几种循环的比较……（48）
2.3.6　循环嵌套 ……（49）
2.4　break 语句和 continue 语句 ……（50）
2.4.1　break 语句……（50）
2.4.2　continue 语句……（51）
2.5　goto 语句……（52）
2.6　指针程序设计 ……（53）
2.6.1　指针 ……（53）
2.6.2　指针变量的使用……（54）
2.7　典型例题……（56）
习题……（61）
第 3 章　模块化程序设计 ……（67）
3.1　模块化程序设计思想……（67）
3.2　函数定义 ……（69）
3.3　函数调用 ……（70）
3.3.1　函数调用的形式……（70）
3.3.2　函数间的参数传递……（71）
3.3.3　函数的返回值……（73）
3.4　函数的原型与声明 ……（73）
3.5　函数的嵌套与递归 ……（75）
3.5.1　函数的嵌套调用……（75）
3.5.2　函数的递归调用……（76）
3.6　库函数……（77）
3.7　变量的作用域与存储类型……（78）
3.7.1　变量的作用域……（78）

3.7.2　变量的存储类型……（81）
3.8　指针与函数……（84）
3.8.1　指针作为函数参数……（84）
3.8.2　返回指针值的函数……（87）
3.8.3　指向函数的指针变量……（88）
3.9　典型例题……（89）
习题……（93）
第 4 章　简单构造数据类型……（98）
4.1　一维数组……（98）
4.1.1　一维数组的引出……（98）
4.1.2　一维数组的定义和引用……（99）
4.1.3　一维数组程序举例……（100）
4.2　二维数组……（102）
4.2.1　二维数组的引出……（102）
4.2.2　二维数组的定义和引用……（103）
4.2.3　二维数组程序举例……（104）
4.3　字符数组与字符串……（106）
4.3.1　字符数组的引出……（106）
4.3.2　字符数组的定义和引用……（107）
4.3.3　字符串的使用……（108）
4.3.4　字符数组程序举例……（112）
4.4　数组与指针……（114）
4.4.1　一维数组与指针……（114）
4.4.2　多维数组与指针……（116）
4.4.3　数组作为函数参数……（117）
4.5　字符串与指针……（121）
4.5.1　用字符指针访问字符串……（121）
4.5.2　字符指针和字符数组的区别……（122）
4.5.3　字符串作为函数参数……（124）
4.6　典型例题……（125）
习题……（132）
第 5 章　复杂构造数据类型……（139）
5.1　结构体……（139）
5.1.1　结构体的引出及使用……（139）
5.1.2　结构体数组……（143）
5.1.3　结构体程序举例……（145）
5.1.4　结构体与指针……（146）
5.2　共用体……（149）
5.2.1　共用体的定义和引用……（149）
5.2.2　共用体类型的特点……（150）
5.2.3　共用体应用举例……（151）
5.3　枚举类型……（152）
5.3.1　枚举类型的定义和引用……（152）
5.3.2　枚举类型应用举例……（153）
*5.4　链表……（154）
5.4.1　概述……（154）
5.4.2　简单链表……（155）
5.4.3　动态链表……（155）
5.4.4　链表的实现及应用……（156）
习题……（159）
第 6 章　磁盘数据存储……（163）
6.1　将数据写入文件……（163）
6.1.1　打开文件函数……（164）
6.1.2　关闭文件函数……（164）
6.2　文件读写分类函数……（165）
6.2.1　单字符写入函数……（166）
6.2.2　单字符读取函数……（166）
6.2.3　字符串读取函数……（167）
6.2.4　字符串写入函数……（168）
6.2.5　格式化读写函数……（168）
6.2.6　数据块读写函数……（170）
6.3　文件定位函数……（172）
6.3.1　位置指针复位函数……（172）
6.3.2　位置指针的随机移动函数……（174）
6.3.3　文件指针当前位置函数……（175）
6.4　其他文件函数……（176）
6.4.1　文件结束检测函数……（176）
6.4.2　出错检测函数……（177）
习题……（177）
第 7 章　实用程序设计技巧……（181）
7.1　程序的模块化结构……（181）
7.1.1　软件工程的思想……（181）
7.1.2　模块设计……（181）
7.1.3　模块化的优点……（182）
7.2　模块的组装……（183）

7.2.1　文件包含与头文件的使用 ……（183）
7.2.2　模块间的连接 ……（185）
7.2.3　标识符的一致性……（188）
7.2.4　条件编译 ……（188）
7.3　模块设计风格简述 ……（190）
7.3.1　数据风格 ……（190）
7.3.2　标识符风格 ……（190）
7.3.3　算法风格 ……（191）
7.3.4　输入/输出风格 ……（191）
7.3.5　书写风格 ……（192）
7.4　应用程序设计实例 ……（193）
习题 ……（200）
附录 ……（203）
附录 A　常用 C 语言库函数 ……（203）
A.1　数学函数 ……（203）
A.2　输入/输出函数……（203）
A.3　字符函数 ……（206）
A.4　字符串函数 ……（207）
A.5　动态存储分配函数……（207）
A.6　时间函数 ……（208）
A.7　其他函数 ……（209）
附录 B　ASCII 码表 ……（210）
附录 C　C 语言运算符的优先级与结合性 ……（212）
参考文献 ……（214）

第 1 章　C 语言程序基础

计算机之所以能够自动、高速地进行大量的计算、处理各种信息，源于事先存储的程序。计算机的所有操作都是在程序的控制之下完成的，计算机离不开程序。程序是为实现特定目标或解决特定问题而用计算机语言编写的命令序列的集合。

C 语言是计算机最常用的程序设计语言，是国际上最流行的、应用面最广的高级程序设计语言。C 语言也是一种结构化程序设计语言，用它编写的程序层次清晰，便于按模块化方式组织，易于调试和维护。40 多年来，C 语言经历了不断的发展和完善，已经成为国内外公认的一种优秀的程序设计语言，有着其他语言不可比拟的优点。

本书以 C 语言程序设计为主线，介绍程序设计的基本概念和基本方法。

1.1　C 语言程序的基本结构

1.1.1　认识 C 语言程序

为说明 C 语言程序的结构，首先来看下面简单的 C 语言程序。

【例 1.1】 在屏幕上输出一行信息：This is a C program.

```
#include <stdio.h>                      /* 预处理命令 */
int main( )                             /* 定义主函数 */
{
    printf("This is a C program.\n");   /* 调用库函数，输出信息 */
    return 0;                           /* 返回 0 */
}
```

【例 1.2】 计算两数之和，并输出结果。

```
#include <stdio.h>                      /* 预处理命令 */
int main( )                             /* 定义主函数，计算两数之和 */
{
    int a,b,sum;                        /* 定义三个整型变量 */
    a=123; b=456;                       /* 为变量 a，b 赋值 */
    sum=a+b;                            /* 让 sum 等于 a+b 的值 */
    printf("sum=%d\n",sum);             /* 调用库函数，输出结果 */
    return 0;                           /* 返回 0 */
}
```

现在对这两个 C 语言程序进行解析。

① #include <stdio.h>是预处理命令，表示文件包含，其功能是将头文件 stdio.h 的内容包含到用户源程序中。文件 stdio.h 中声明了程序所需要的输入和输出操作的有关信息，有了该预处理命令后，程序中就可以使用包含文件的内容了。

② main 表示主函数，main 是函数名。每个 C 语言程序必须有 main 函数。函数名后的一对圆括号不能省略。一个 C 语言程序可以包含若干个函数，但只能有一个主函数。main 前面的 int 是一种数据类型，说明函数执行完毕后向系统返回一个整数，具体返回的数值即 return 语句后的 0，返回 0 表示程序正常结束。

③ 用{ }括起来的是 main 的函数体。main 函数中的所有操作（语句）都在这一对{ }之间。也就

是说 main 函数的所有操作都在 main 函数体中。

④ printf 是 C 语言的标准输出函数，其含义是将双引号内的内容输出到显示器屏幕上。例 1.1 中的 printf 用于将字符串“This is a C program.\n”输出，即在屏幕上显示：This is a C program.后光标跳到下一行行首。“\n”是换行符，使光标跳到下一行行首（第一列）。例 1.2 中的%d 为格式控制符，表示在此位置将输出一个十进制整数，该整数由逗号后面的变量 sum 提供。程序运行后，将在屏幕上输出：sum=579。

⑤ 分号“;”是 C 语句结束符，表示该语句结束。

⑥“/*　*/”括起来的部分是一段注释，注释只是为了改善程序的可读性，在编译、运行时不起作用（事实上编译时会跳过注释，目标代码中不会包含注释）。注释可以放在程序的任何位置，并允许占用多行，只是需要注意“/*”与“*/”匹配且不能嵌套注释。在 Visual C++集成开发环境下，注释可写在“//”后，但注释内容仅限“//”所在行。

⑦“int a,b,sum;”是变量声明，声明了 3 个具有整数类型的变量 a、b、sum。C 语言的变量必须先声明后使用。

⑧“a=123; b=456;”是两条赋值语句，将 123 赋给变量 a，456 赋给变量 b。执行上述两条语句后，a、b 两个变量的值分别为 123 和 456。

⑨“sum=a+b;”将 a、b 两个变量的内容相加，然后将结果赋值给整型变量 sum。此时 sum 的内容为 579。

【例 1.3】 输入两个整数，计算并输出两者中较大的数。

```
#include <stdio.h>
int max(int x,int y)                    // 定义 max 函数，用于计算两数中较大的数
{                                       // max 函数体开始
    int z;                              // 声明部分，定义变量
    if (x>y) z=x;                       // 如果 x>y，则将 x 的值赋给 z
    else z=y;                           // 否则，则将 y 的值赋给 z
    return z;                           // 将 z 值返回，通过 max 带回调用处
}                                       // max 函数体结束
int main( )                             // 主函数
{                                       // main 函数体开始
    int a,b,c;                          // 声明部分，定义变量
    scanf("%d%d",&a,&b);                // 调用库函数，输入变量 a、b 的值
    c=max(a,b);                         // 调用 max 函数，将调用结果赋给 c
    printf("max=%d\n",c);               // 调用库函数，输出 c 的值
    return 0;                           // 返回 0
}                                       // main 函数体结束
```

说明：

① 本程序包括两个函数。其中，主函数 main 仍然是整个程序执行的起点，函数 max 的功能是计算两数中较大的数。

② scanf 是 C 语言的标准输入函数，用于从键盘输入若干数据给指定变量。%d 表示输入十进制整数。主函数 main 调用 scanf 函数从键盘获得两个整数，存入 a、b 两个变量中，然后调用函数 max 获得两个数中较大的值，并赋给变量 c，最后输出变量 c 的值（结果）。

③ int max(int x,int y)是函数 max 的函数首部，表明此函数从调用它的函数（此处为 main 函数）获得两个整数，返回一个整数。

④ 用{ }括起来的部分是 max 函数的函数体。max 的函数体是函数 max 的具体实现，它从参数表获得数据，处理后得到结果 z，然后将 z 返回调用函数 main。

⑤ 本例表明，除了可以调用库函数外，还可以调用用户自己定义、编写的函数。

1.1.2　基本结构

综合上述 3 个例子，下面对 C 语言程序的基本组成和形式（程序结构）加以说明。

C 语言程序的结构如下：

```
预处理命令
int main( )                    // 主函数
{                              // 函数体开始
    声明部分
    执行部分
    return 0;
}                              // 函数体结束
其他函数                       // 自定义函数
{
    声明部分
    执行部分
}
```

下面对 C 语言程序的结构进行简单说明。

1．函数是 C 语言程序的基本单位

一个 C 语言源程序必须包含一个 main 函数，还可以包含一个或多个其他函数。函数是 C 语言程序的基本单位。C 语言是函数式的语言，程序的全部工作都是由各个函数完成的。编写 C 语言程序就是编写一个个函数。

程序中的函数有些是 C 语言的标准库提供的，称为标准函数（或库函数），如 printf 函数和 scanf 函数。编写程序时，如果有标准函数，就使用标准函数；如果没有标准函数，则需要用户自己编写函数。用户编写的函数可以写在 main 函数的前面或后面，可由 main 函数调用，也可以被其他自定义函数调用。通常，每个函数都完成一项独立的功能。

C 语言函数库非常丰富，ANSI C 提供 100 多个库函数，Turbo C 提供 300 多个库函数，而 Visual C++提供的库函数更多。

不同的编译系统除了提供函数库中的标准函数外，还按照硬件的情况提供一些专门的函数，因此不同编译系统提供的函数数量和功能会有一定差异。

2．main 函数

主函数是每个程序执行的起点。每个 C 语言程序都是从 main 函数开始执行的，而不论 main 函数在程序中的位置。可以将 main 函数放在整个程序的最前面，也可以放在整个程序的最后，或者放在一些函数之后另一些函数之前。但一个程序只能有一个 main 函数。

3．函数的结构

无论是 main 函数还是自定义函数，每个函数都由函数首部和函数体两部分组成。

（1）函数首部

即一个函数的第一行，格式如下：

返回值类型　函数名([函数参数类型 1　函数参数名 1], …, [函数参数类型 *n*　函数参数名 *n*])

例如：

```
int max(int x,int y)
```

注意：一个函数可以没有参数，但是后面的一对括号不能省略，这是格式的规定，如 main 函数。

（2）函数体

函数首部下面用一对{ }括起来的部分。如果函数体内有多对{ }，则最外层是函数体的范围。函

数体一般包括声明部分和执行部分。

- 声明部分：定义本函数所使用的变量，并为每个变量分配相应大小的内存单元。变量名是内存单元的符号地址。
- 执行部分：由若干条语句组成的命令序列（可以在其中调用其他函数）。

4．书写风格

C 语言程序书写格式自由，一行可以写多个语句，一个语句也可以写在多行上。每条语句的最后必须在末尾用分号“;”表示语句的结束。

5．输入/输出

C 语言本身不提供输入/输出语句，输入/输出操作是通过调用库函数（如 scanf 和 printf 等）完成的。

由于输入/输出操作涉及具体的计算机硬件，因此把输入/输出操作放在函数中处理可以简化 C 语言和 C 语言的编译系统，便于 C 语言在各种计算机上实现。

6．注释

注释部分要放在符号“/*”和“*/”之间或者“//”之后并与其同行。为了便于阅读，程序中应适当地加入注释。注释只起帮助阅读和理解程序的作用，不参加程序的编译，也不影响程序的运行。使用注释是编程人员的良好习惯。

实践中，编写的程序往往需要不断地修改和完善，事实上没有一个应用程序是不需要修改和完善的。很多人会发现自己编写的程序在经历了一些时间以后，如果缺乏必要的文档和必要的注释，最后连自己都很难再读懂，需要花费大量时间重新思考和理解原来的程序，因而浪费了大量的时间。如果一开始编程就对程序进行注释，虽然编写程序时麻烦一些，但日后可以节省大量的时间。

7．预处理命令

预处理命令能够改进程序的设计环境，提高编程效率。C 语言的预处理功能主要包括：宏定义、文件包含和条件编译，分别用宏定义命令（#define）、文件包含命令（#include）和条件编译命令（#ifdef…#else…#endif）实现，为了与一般语句区别，这些命令以“#”开头。

1.1.3　程序举例

【例 1.4】 试编写程序，计算圆的周长和面积。

分析：计算圆的周长和面积要用到圆周率π，它的值为 3.14，但在 C 语言程序中不能直接使用希腊字母π，可以直接使用 3.14，或定义一个常量或变量来表示π，本程序中定义了一个变量 PI，令 PI=3.14。

已知圆的半径为 r，则周长 l=2*PI*r，面积 s=PI*r*r。因为 PI 包含小数，所以变量应定义为可以存储小数且精度较高的双精度型变量 double。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    double PI=3.14,r,l,s;                    // 声明 double 型变量
    printf("Input r:\n");                    // 输入提示
    scanf("%lf",&r);                         // 输入半径 r
    l=2*PI*r;                                // 计算周长，注意与数学表达式的区别
    s=PI*r*r;                                // 计算面积
    printf("%.2f,%.2f\n",l,s);               // 以 2 位小数形式输出
```

```
    return 0;                                          // 返回 0
}
Input r:
1.5
9.42, 7.07
```

【例 1.5】 已知 a=5，b=10，试交换 a、b 的值。

分析：交换两个变量的值，就像一个瓶装酱油，一个瓶装醋，要交换需要借助第三个空瓶一样，交换两个数也需要借助第三个变量。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int a=5,b=10,temp;                 // 定义变量 a、b 和 temp，并对 a、b 赋初值
    temp=a;                            // 将变量 a 的值赋给 temp 保存
    a=b;                               // b 的值赋给 a
    b=temp;                            // temp 的值赋给 b
    printf("a=%d,b=%d\n",a,b);         // 输出 a、b 的值
    return 0;
}
```

运行结果：

```
a=10,b=5
```

【例 1.6】 试计算圆柱体的表面积。

分析：为了计算圆柱体的表面积，可以利用公式 $2\pi r$（r 为半径）得到底面周长 d，利用公式 $d\times h$ 计算侧面积 s_1，利用公式πr^2 计算底面积 s_2，则表面积 $s=s_1+2\times s_2$。因此可以按照以下步骤计算：

① 设圆柱体的高为 h，半径为 r，表面积为 s；

② 输入 r、h 的值；

③ 计算底面周长 $d=2\pi r$；

④ 计算侧面积 $s_1=dh$；

⑤ 计算底面积 $s_2=\pi r^2$；

⑥ 计算表面积：$s=s_1+2\times s_2$；

⑦ 输出 s 的值。

这是用自然语言描述的计算圆柱体表面积的算法。在计算过程中，有些步骤可以合并。现将其转换为 C 语言程序。注意变量 r、h 和 s 的数据类型。

参考程序如下：

```
#include <stdio.h>                         // 预处理文件包含命令
#define PI 3.1415926                       // 预处理，定义符号 PI 代表常量 3.1415926
int main( )                                // main 函数
{
    double r,h,s;                          // 声明部分：定义 3 个变量 r,h,s，都为实数类型
    printf("Input r,h:\n");                // 执行部分：输出提示信息
    scanf("%lf%lf",&r,&h);                 // 执行部分：输入 r 和 h 的值
    s=2*PI*r*h+2*PI*r*r;                   // 执行部分：计算表面积
    printf("Total area is %.2f\n",s);      // 执行部分：输出表面积
    return 0;
}
```

说明：

①“#define PI 3.1415926”是宏定义命令，PI 称为符号常量，即用一个标识符代表一个常量。此处表示用标识符 PI 代表常量 3.1415926。

符号常量的定义形式为：

#define 标识符 常量

② double 表示实型，即定义 r、h 和 s 为实型变量。double 型数据在输入时需要用%lf 格式控制符，输出时可用%f 或%lf 格式控制符，“.2”表示只输出 2 位小数。

1.2　C 语言基本语法成分

1．C 语言字符集

字符是 C 语言最基本的元素，C 语言字符集由字母、数字、空白、标点符号和特殊字符组成（在字符串常量和注释中还可以使用汉字和其他图形符号）。由字符集中的字符可以构成 C 语言进一步的语法成分，如标识符、关键字、运算符等。

C 语言程序是用下列字符所组成的字符集写成的：

① 字母：A～Z，a～z

② 数字：0～9

③ 标点符号、特殊字符：

! # % ^ & + − * / = ~ < > \

| . , ; : ? ' " ( ) [ ] { }

④ 空白符：空格、制表符（Tab 跳格键）、换行符（空行）的总称。空白符除了在字符和字符串中有意义外，编译系统忽略其他位置的空白符。空白符在程序中只是起到间隔的作用。在程序的恰当位置使用空白符将使程序更加清晰，增强程序的可读性。

2．标识符

标识符是用来标识变量名、符号常量名、函数名、数组名、类型名等实体（程序对象）的有效字符序列。标识符由用户定义。

（1）C 语言标识符定义规则

① 标识符只能由字母、数字和下画线 3 种字符组成，且第一个字符必须为字母或下画线。

例如，合法的标识符：a，i，sum，average，_total，Class，day，student，p405；不合法的标识符：5a，M.D.John，$123，3D64，a−b。

② 大小写敏感，即 C 语言程序认为大小写字母是不同的字符。例如，sum 不同于 Sum，BOOK 不同于 book。一般情况下习惯将变量名小写，常量名大写，但不绝对。

③ ANSI C 没有限制标识符长度，但各个编译系统都有自己的规定和限制（TC 允许用 32 个字符，MS C 只能用 8 个字符）。

例如，student_name 和 student_number，在 MS C 中，系统认为这两个标识符是相同的。

④ 标识符不能与“关键字”同名，也不能与系统预先定义的“标准标识符”同名，如 main、printf 等。

（2）定义标识符需遵循的原则

① 尽量不要用下画线开头。因为系统内部使用了一些下画线开头的标识符（如_fd、_cs、_ss），要避免与系统定义的标识符冲突。

② 尽量做到“见名知义”，如 sum，area，score，day，name，age 等。

③ 在容易出现混淆的地方应尽量避免使用容易认错的字符。例如，数字 1、字母 l、字母 I，数字 0、字母 o、O，数字 2、字母 Z、字母 z。

3．关键字

关键字是 C 语言预先定义的、具有特定意义的标识符，也称为保留字。C 语言包括 32 个关键字：

auto	break	case	char	const	continue
default	do	double	else	enum	extern
float	for	goto	if	int	long
register	return	short	signed	sizeof	static
struct	switch	typedef	union	unsigned	void
volatile	while				

C 语言的关键字都是小写的。不能重新定义关键字，也不能把关键字定义为一般标识符。

4．运算符

运算符是用于描述某种运算功能的符号，如+、−、*、/、%等，可以由一个或多个字符组成。由运算符将常量、变量和函数调用连接起来的式子称为表达式。

根据参与运算的操作数个数，运算符分为：单目（一元）运算符、双目（二元）运算符和三目（三元）运算符。

C 语言运算符分为以下几类：

- 算术运算符：+　−　*　/　%
- 自增、自减运算符：++　−−
- 关系运算符：<　<=　>　>=　==　!=
- 逻辑运算符：!　&&　||
- 位运算符：<<　>>　~　|　^　&
- 赋值运算符：=及扩展赋值运算符
- 条件运算符：?:
- 逗号运算符：,
- 指针和地址运算符：*　&
- 求字节运算符：sizeof
- 成员运算符：.　->
- 下标运算符：[ ]
- 强制类型转换运算符：(类型)

部分运算符功能简介如下。

（1）算术运算符

C 语言基本的算术运算符共有 5 种：

＋（加）、−（减）、*（乘）、/（除）、%（取余，模运算）

它们都是双目运算符，即运算符要求有两个操作数。

- 取余运算“%”左右的两数必须为整型数据，如 7%4 值为 3。
- 两个整数相除，结果为整数，舍去小数部分，如 5/3=1。但是，如果除数或被除数中有一个为负值，则舍入的方向是不固定的。如−5/3 在有的机器上得到的结果是−1，有的机器则是−2。多数机器采取“向零取整”的方法，即取整后向零靠拢（即向实数轴的原点靠拢）。
- 字符型数据可以和数值型数据混合运算。因为字符型数据在计算机内部是用一个字节的整型数表示的，如'A'+32。
- 运算符是有优先级高低之分的，遵循的原则是“先乘除，后加减”。“*、/、%”为同一级别，“+、−”为同一级别，若一个运算对象两侧的运算符优先级别相同时，则按规定的“自左至

右”的方向运算。

（2）自增、自减运算符

C 语言的自增、自减运算符只能作用于变量，其作用是使变量的值增 1 或减 1，例如：

++i,—i　　（在使用 i 之前，i 值先加/减 1）

i++,i—　　（在使用 i 之后，i 值再加/减 1）

例如：a=5; b=a++;则 b 取值为 5，但赋值语句结束后 a 值为 6；而如果 b=++a;则赋值语句结束后，a、b 的值都为 6。

说明：++、—运算符是同级优先关系，优先级都高于算术运算符；同时出现时，自增、自减运算符按从右至左的结合方向。

5．分隔符

就像写文章要有标点符号一样，写程序也要有一些分隔符。在 C 语言程序中，空格、逗号、回车/换行符等，在各自不同的应用场合起着分隔符的作用。例如，在程序中的相邻保留字、标识符之间应有空格或回车/换行作为分隔符，相邻同类项之间用逗号作为分隔符。

例如，语句“int x,y;”中的空格和逗号都起着分隔符的作用。如果缺少了分隔符，程序就会出错。例如，缺少了逗号，x 和 y 就会被认为是一个变量 xy。

6．其他符号

花括号“{”和“}”通常用于标识函数体或一个语句块（即复合语句）。

“/*”和“*/”构成一组注释符，它们之间的内容表示注释，常用于多行注释。“//”表示自它开始直到行末均为注释，常用于单行注释。编译时编译系统忽略注释。

① 注释在程序中起提示、解释作用。

注释与软件的文档同等重要，要养成使用注释的良好习惯，这对软件的维护相当重要。请记住，程序是要给别人看的，自己也许还会看自己几年前编制的程序，清晰的注释有助于他人理解程序、算法的思路。

② 在软件开发过程中，还可以将注释用于程序的调试，暂时屏蔽一些语句。

例如，在调试程序时暂时不需要运行某段语句，而又不希望立即从程序中删除它们，可以使用注释符将这段程序框起来，暂时屏蔽这段程序，以后可以方便地恢复。

1.3　C 语言数据类型

为了按不同的方式和要求处理数据，数据必须区分为不同的类型。在 C 语言程序中，每个数据都属于一个确定的、具体的数据类型。不同的数据类型在数据表示形式、合法的取值范围、占用内存空间大小、可参与的运算种类等方面都有所不同。

C 语言的数据类型十分丰富，如图 1.1 所示。

C 语言中的数据有常量与变量之分，它们分别属于上述不同的类型。常量与变量的区别是：在程序执行过程中，常量的值是固定的，不能改变；变量的值存储在一个由变量名标识的内存存储区中，其值可由程序改变，因此每个变量都有一个名字，

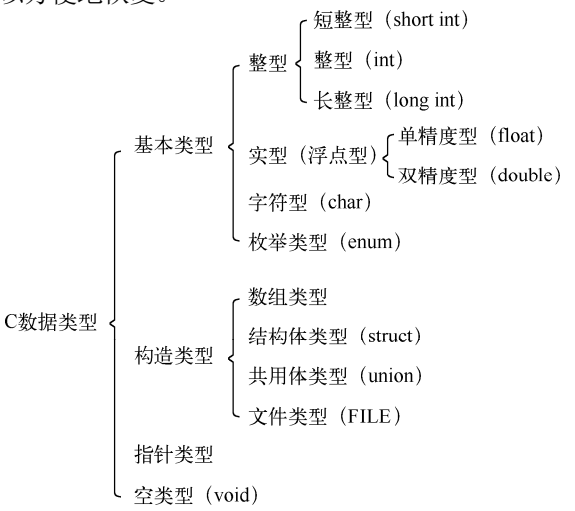

图 1.1　C 语言的数据类型

如 x，sum，area 等。

变量在使用前必须进行说明，其目的是为变量在内存中申请存放数据的内存空间。

1.3.1　基本数据类型

1．整型数据

（1）整型常量

整型常量即整型常数。在 C 语言中，整型常量有 3 种表示方式：

① 十进制数：以非 0 数字开头的数，如 123、−5、0。

② 八进制数：以数字 0 开头的数，如 0123 表示八进制数$(123)_8$，等于十进制数 83。

③ 十六进制数：以 0x 或 0X 开头的数，如 0x123 表示十六进制数$(123)_{16}$，等于十进制数 291。

（2）整型变量

C 语言中的整型变量分为 short int、int 和 long int 三种类型，每种类型又分为有符号类型和无符号类型，分别用 signed 和 unsigned 表示，默认为 signed 类型。各种整型变量的长度和表示范围如表 1.1 所示。

表 1.1　整数类型

类　型	Turbo C 比特数	取 值 范 围	Visual C++ 6.0 比特数	取 值 范 围
[signed] int	16	−32768～32767 即-2^{15}～（$2^{15}-1$）	32	−2147483648～2147483647 即-2^{31}～（$2^{31}-1$）
unsigned [int]	16	0～65535 即 0～（$2^{16}-1$）	32	0～4294967295 即 0～（$2^{32}-1$）
[signed] short [int]	16	−32768～32767 即-2^{15}～（$2^{15}-1$）	16	−32768～32767 即-2^{15}～（$2^{15}-1$）
unsigned short [int]	16	0～65535 即 0～（$2^{16}-1$）	16	0～65535 即 0～（$2^{16}-1$）
long [int]	32	−2147483648～2147483647 即-2^{31}～（$2^{31}-1$）	32	−2147483648～2147483647 即-2^{31}～（$2^{31}-1$）
unsigned long [int]	32	0～4294967295 即 0～（$2^{32}-1$）	32	0～4294967295 即 0～（$2^{32}-1$）

提示：在流行的编译系统中，int 型数据的长度都采用 32 位，同 long int，本书均采用 32 位。

如果在整常数后面加字母 U 或 u，表示 unsigned int 型常数；加字母 L 或 l，表示 long int 型常数。

变量的定义形式为：

```
数据类型 变量名表;
```

其中，变量名表的变量可为 1 个或多个，中间用逗号“,”分隔。

例如：
```
int a,b,sum;
short int x;
```

2．实型数据

带有小数点的数称为实数，又称为浮点数。

（1）实型常量

C 语言中，实型常量有两种表示方式。

① 十进制数形式：由数字和小数点组成（必须有小数点），但小数点前后的 0 可以省略，如 0.12、.12、123.0、123.、0.等。

② 指数形式：例如 1.2e3 和 0.12E4 都表示 1.2×10^3。注意，e（或 E）前面要有数字，且 e（或

E）后面的指数必须为整数。例如 2.3e3.5，e2 等都是不合法的表示形式。

（2）实型变量

C 语言中的实型变量分为单精度型（float）、双精度型（double）和长双精度型（long double），各种类型的长度和数值范围如表 1.2 所示。

表 1.2　实数类型

类　型	比特数	有效数字	数值范围
float	32	6～7	-3.4×10^{38}～3.4×10^{38}
double	64	15～16	-1.7×10^{308}～1.7×10^{308}
long double	80	18～19	-3.4×10^{4932}～1.1×10^{4932}

一个实型常数默认为 double 型。要表示 float 型数，则必须在实数后加字母 f 或 F。要表示 long double 型数，则必须在实数后加字母 l 或 L。

提示：Visual C++ 6.0 中把 long double 等同于 double。

3．字符型数据

字符型数据是用来存放 ASCII 码字符的。每个字符型数据占 1 个字节的内存单元，因此 1 个字符型数据中，只能存放 1 个字符。每个字符都对应 1 个 ASCII 码值，在内存中实际存放的是该字符的 ASCII 码值。常见字符的 ASCII 码表见附录 B。

（1）字符常量

字符常量是用单引号（'）括起来的单个字符，如'A'、'a'、'0'、'$'、','等。

可用字符常量为字符变量赋值，例如：

```
char ch;
ch='A';
```

以上语句定义了字符变量 ch，然后给变量 ch 赋值为字符'A'，当然也可以直接用字符'A'的 ASCII 值赋值，如：“ch=65;”。

注意：

ASCII 码表中数字字符都对应有相应的 ASCII 码值，例如：数字字符'0'的 ASCII 码值为 48，因此在为字符变量赋值的时候应注意书写规则。“ch='0';”与“ch=0;”代表赋予变量不同的字符，前者是用字符常量形式表示的数字字符 0，而后者表示以 ASCII 码值的形式赋值，对应的 ASCII 值为 0 的字符是空操作字符（即字符串常量的结束标志'\0'）。

C 语言中，还有一种特殊的字符常量，即以“\”开头的字符序列，称为转义字符。转义字符将反斜杠“\”后面的字符转变成另外的意义，它通常用来表示 ASCII 码字符集中一些特定的、具有控制功能的字符，如'\n'。表 1.3 中列出了 C 语言中常用的转义字符。

表 1.3　常用的转义字符

字符形式	ASCII 码	功　能
\a	7	响铃
\n	10	换行
\t	9	制表符（横向跳格），光标跳到下一输出区（每个输出区占 8 列）
\v	11	竖向跳格
\b	8	退格
\r	13	回车但不换行，即使光标移到本行第一列
\\	92	反斜杠字符“\”
\"	34	双引号
\'	39	单引号

续表

字 符 形 式	ASCII 码	功　　能
\0	0	字符串结束标志
\ddd		用 1～3 位八进制数表示的任意字符，如'\101'为'A'
\xhh		用 1～2 位十六进制数表示的任意字符，如'\x41'为'A'

（2）字符变量

字符变量是用来存放字符常量的。每个字符变量占 1 个字节的内存单元，在 1 个字符变量中，只能存放 1 个字符常量。

字符变量的定义形式如下：

```
char ch1,ch2;
```

注意：

① 大多数编译系统中，将 char 型数据作为有符号类型（signed）处理，即表示范围为−128～+127，如果要表示 128～255 范围内的字符，需要定义为 unsigned 类型，见表 1.4。

② '0'与 0 是截然不同的两个数。'0'是数字字符，其 ASCII 码值为 48（0x30），而 0 则是整数值，表示 ASCII 为 0 的字符（即'\0'）。

表 1.4　字符类型

类　　型	比 特 数	取 值 范 围
[signed] char	8	−128～+127
unsigned char	8	0～255

（3）字符串常量

字符串常量是由一对双引号（" "）括起来的字符序列。例如，"Hello"、"I love China!"、"How do you do?"等都是字符串常量。

C 语言未对字符串的长度加以限制。为了判断字符串结束，C 语言编译器会自动在字符串的末尾加一个转义字符'\0'，作为字符串常量的结束标志。因此，字符串"China"在内存中占 6 个字节的连续内存单元，如图 1.2 所示。

C	h	i	n	a	\0

图 1.2　字符串在内存中的存放

另外，C 语言中没有专门的字符串变量，而将字符串常量存放在字符型数组中（将在第 4 章中介绍）。

（4）字符型数据与整型数据的相互运算

由于整型数据在内存中是以二进制补码形式存放的，字符型数据在内存中以相应的 ASCII 码值存放，实际上也是以二进制数形式存放的。因此，C 语言允许字符型数据与值在 0～127 范围内的整型数据之间可以通用。

4．枚举类型

当一个变量的取值仅有有限几种可能时，可以使用枚举类型。枚举类型属于简单数据类型，但其使用方法与结构体类型相似。我们将在复杂构造数据类型中讲述其使用方法。

1.3.2　指针类型

指针是一种特殊的数据类型，存储的数值被解释成为内存里的一个地址。要理解指针的含义，必须清楚变量在内存中的存储。

计算机内存是由一片连续的存储单元组成的，操作系统给每个内存单元（一般以字节为单位）一个编号，这个编号称为内存单元的地址（简称地址），每个存储单元占用内存的 1 个字节。在运行程序时，变量的值一般存储在计算机的内存中，变量通过内存地址存取变量的值。寻找地址的过程由系统完成，对用户是不可见的。不同类型的数据占据的内存单元数是不同的，如前面讲的基本数据类型，

int 类型变量在 Visual C++ 6.0 环境下占 4 个字节的连续空间，double 类型变量占 8 个字节的连续空间。在存取数据时，变量的地址是第一个存储单元的地址，系统根据变量的类型确定存取的内存单元数。既然变量通过内存地址存取数据，那么如果知道变量的内存地址，就可以不使用变量名也能存取数据。

变量的地址也叫变量的指针，是一种指针常量。变量的值和变量的地址是不同的概念，变量的值是该变量在内存单元中的数据。变量、内存地址、变量值的关系如图 1.3 所示。

指针变量是一类存取其他变量或函数的地址的变量。与其他变量所不同的是，一般的变量包含的是实际的、真实的数据，而指针变量是一个指示器，它告诉程序在内存的哪块区域可以找到数据。

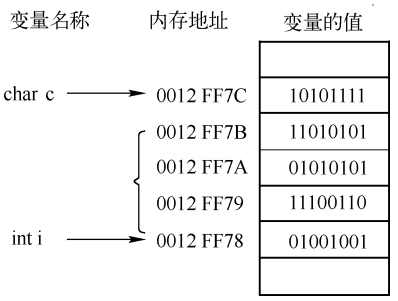

图 1.3　变量、内存地址、变量值的关系

1.3.3　构造数据类型

构造数据类型是由一种或多种数据类型通过某种方式构造而成的，有关构造数据类型将在以后有关章节中介绍。

1.4　C 语言的表达式和语句

由运算符将常量、变量和函数调用连接起来的式子称为表达式。

表达式语句是在表达式的最后加上一个“;”构成的，分号是语句不可缺少的一部分。C 语言程序中大多数的语句都是表达式语句。表达式语句常见的形式有：赋值语句、函数调用语句和空语句。

1．赋值语句

赋值语句由赋值表达式加一个分号构成。

C 语言的赋值语句先计算赋值运算符（=）右边的子表达式的值，然后将此值赋值给左边的变量。

对变量赋值的一般格式如下：

```
<变量>=<表达式>
```

例如：

```
b=30;                          // 将表达式的值 30 赋给变量 b
a=sin(b*3.14159/180);          // 将表达式（正弦函数）的值赋给变量 a，即求 sin(π/6)
```

变量赋值的特点如下。

① 变量必须先定义，后使用。例如，“int d,e,f;”定义了 3 个变量为整数类型。如果变量未定义，则在编译时被认为是非法的，提示错误。

② 变量被赋值前，值不确定。

③ 对变量的赋值过程是“覆盖”过程，即用新值去替换旧值。

④ 读出变量的值，变量的值保持不变。

⑤ 参与表达式运算的所有变量都保持原来的值不变。

⑥ 赋值运算是右结合性，即“取右送左”。若有声明语句：

```
int a=1,b=2,c=3;
```

则执行表达式：

```
a=b=c;
```

结果为 a、b、c 的值都是 3（先运算 b=c，再运算 a=b）。

2．函数调用语句

函数调用语句由函数调用表达式加一个分号构成。例如：

```
printf("This is a C statement.");                    c=max(a,b);
```

3．空语句

空语句即只有一个分号的语句，它什么也不做（表示这里需要一个语句，但是不需要做任何工作）。

4．复合语句

复合语句是由花括号将多条语句组合在一起构成的，常用于流程控制语句中执行多条语句，在语法上相当于一条语句。复合语句的形式为：

```
{
    [内部变量声明;]        // 可以在复合语句内部定义变量，见第 3 章
    数据操作语句 1;
    …
    数据操作语句 n;
}
```

例如：

```
int a,b;
if (a<b)
{
    int temp;                    // 复合语句内部定义的变量
    temp=a;
    a=b;
    b=temp;
}
```

使用复合语句时应注意：

① 在复合语句的“内部变量定义语句”中定义的变量，是局部变量（详见第 3 章），仅在复合语句中有效。上例中 temp 在“{}”外无效。

② 复合语句结束的“}”之后，不需要再加分号。

1.5　C 语言程序运行过程

1.5.1　源程序、目标程序和可执行程序的概念

1．程序

程序是一组计算机可以识别和执行的指令，每一条指令使计算机执行特定的操作。

2．源程序

程序可以用高级语言或汇编语言编写，用高级语言或汇编语言编写的程序称为源程序。C 语言源程序的扩展名为“.c”。

源程序不能直接在计算机上执行，需要用“编译程序”将源程序翻译为二进制数形式的代码。

3．目标程序

源程序经过“编译程序”翻译所得到的二进制代码称为目标程序。目标程序的扩展名为“.obj”。

目标代码尽管已经是机器指令，但还不能运行，因为目标程序还没有解决函数调用问题，因此需要将各个目标程序与库函数链接起来，才能形成完整的可执行程序。

4．可执行程序

目标程序与库函数连接，形成完整的、可在操作系统下独立执行的程序称为可执行程序。可执行

程序的扩展名为“.exe”（在 DOS/Windows 环境下）。

1.5.2　C 语言程序的开发步骤

C 语言采用编译方式将源程序转换为二进制目标代码。程序的开发过程包括编辑、编译、链接和执行几个步骤，如图 1.4 所示。

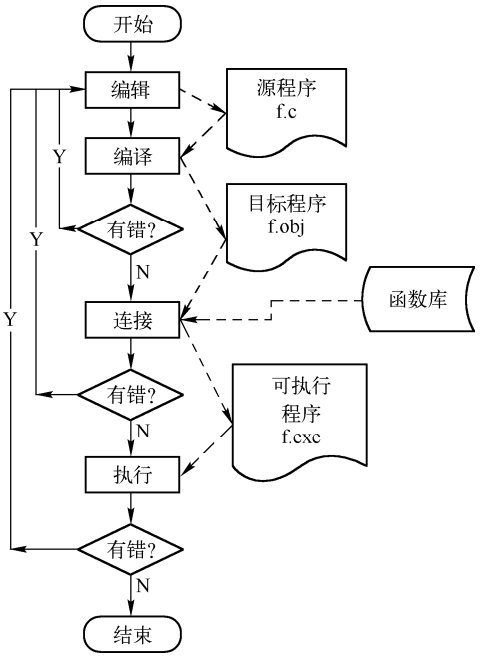

图 1.4　C 语言程序的开发步骤

1．编辑

将编写完成的源程序输入到计算机中，并保存为磁盘文件，扩展名为.c。编辑的对象是源程序，它以 ASCII 代码的形式输入和存储，不能被计算机执行。常用的编辑软件是 Visual C++ 6.0、Codeblocks、Dev C、Turbo C、Borland C 等，也可以使用 Windows 下的记事本等字处理软件。

2．编译

编译具体分为执行预处理和编译两个阶段。

① 执行预处理：如果程序中有预处理命令（如#include 命令或#define 命令），则首先要处理预处理指令，然后进行下面的编译过程；如果源程序中无预处理指令，则直接进行下面的编译过程。

② 编译：编译是将源程序代码翻译为二进制目标代码的形式——目标程序，扩展名为.obj。在编译过程中，如果发现语法错误，屏幕上会显示出错信息并中止编译。此时应对源程序的错误进行修改，重新进行编译，如此反复进行，直到完全正确为止，最后生成目标代码程序。

应当指出，经编译后得到的二进制目标代码是不能直接执行，因为每一个模块往往是单独编译的，必须把经过编译的各个模块的目标代码与系统提供的标准模块（如 C 语言中的标准库函数）连接后才能运行。

3．连接

将各模块的二进制目标代码与系统标准模块连接在一起，生成扩展名为.exe 的可执行文件。

4．执行

执行经过编译和链接的可执行文件。一般在显示器上显示运行结果，可根据运行结果来判断程序是否有错，如果有错误，则必须改正，并重新进行编译和连接，然后运行。这样反复运行，直到得到正确的运行结果。

集成化的工具环境（如 Visual C++、Codeblocks、Dev C、Borland C 和 Turbo C 等）已经将编辑、编译、链接和调试工具集于一身，用户可以方便地在窗口状态下连续进行编辑、编译、链接、调试和运行的全过程。

本书采用 Visual C++ 6.0 集成开发环境。Visual C++ 6.0 是一款基于 Windows 环境运行的可视化面向对象编程工具。C 语言程序在 Visual C++ 6.0 环境下编辑、调试和运行，可提高源程序的编辑效率，方便修改程序，便于初学者使用，同时为将来学习 C++编程奠定一定的基础。当然在 Visual C++ 6.0 中，系统提供的库函数与在 Turbo C 中和 Borland C 中略有不同，如有些数据类型

的长度定义不同，读者在参考其他书籍的例题源程序时应加以注意。

1.6　编写简单的 C 语言程序

程序设计时，通常采用 3 种不同的程序结构，即顺序结构、选择结构和循环结构，其中顺序结构是最基本、最简单的程序结构。现在开始最简单的 C 语言程序设计，即顺序程序设计的学习。

【例 1.7】 编写一个显示基本算术运算功能的程序，包括取余、整除、自增、自减、强制转换、自动转换等。

分析：C 语言的基本算术运算包括两类：

① 单目运算符，如−（取负）、++（自增）、−−（自减）。

② 双目运算符，如*（乘）、/（除）、%（取余，模）、+（加）、−（减）。

单目运算，是指使用一个操作数进行运算，如−5、i++等；双目运算是指使用两个操作数进行运算，如 a+b、3*c、n/m 等。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int i=0,j=0,t=10,x=10,y=10,z=10,b,c;
    double f,f1,f2;
    // 单目运算
    x=−x;                          // 运算结果 x=−10
    y=++i;                         // 运算结果 y=1，i=1，相当于执行 i=i+1; y=i;
    z=j++;                         // 运算结果 z=0，j=1，相当于执行 z=j; j=j+1;
    printf("x=%d,y=%d,z=%d\n",x,y,z);
    printf("i=%d,j=%d\n",i,j);
    // 双目运算
    i=t/3;                         // 整除运算 i=3
    j=t%3;                         // 取余运算 j=1
    printf("i=%d,j=%d\n",i,j);
    // 类型转换
    f=10;
    b=2;
    f1=1/b*100+10+f;               // 1/2 为整型数结果为 0，乘 100 仍为 0，f1=20.0
    f2=1.0/b*100+10+f;             // 1.0/2*100 为实型数结果为 50.0，f2=70.0
    c=(double)1/b*100+10+f;        // 将 1 强制转换为 double 型数；赋值时将 70.0 转换为 70 赋给 c
    printf("f1=%f,f2=%f\n",f1,f2);
    printf("c=%d\n",c);
    return 0;
}
```

说明：

① 单目运算符“–”对数据取负。++i 先对 i 进行加 1 运算，再使 i 参与其他运算。j++是先使 j 参与其他运算，再对 j 进行加 1 运算。必要时可以加改变运算顺序。

②“i=t/3;”参与运算的数都是整型，得到的结果是商的整数部分，舍去小数向 0 取整，结果 i=3。求模运算后（运算符%）得到的是余数。

③“f1=1/2*100+10+f;”由于 1/2 为整型数结果为 0，因此乘 100 仍然为 0。C 语言规定两个整型数运算的结果依然是整型数，即舍去小数部分，但是如果除数或被除数中有一个为负值，则舍入的方向是不固定的，多数机器采用“向 0 取整”的方法（实际上就是舍去小数部分，注意：不是四舍五入）。所以尽管 f1 定义为实型数，但得到的值是整数，结果为 0。那么如何才能得到我们想要的结果呢？

方法 1：将参加运算的变量定义为实型。似乎可以把所有的变量都定义为实型，这样就不会出现错误了。但是，要知道实型数表示的范围远大于整型数，因此在内存中占用的字节数也多，此外，实型数据运算需要的时间也长——这些都不利于程序的优化和执行效率，所以只在需要使用实型数据时才定义。

方法 2：令“f=1.0/b;”。C 语言规定，如果参加+、−、*、/运算的两个数中有一个为实型数，则结果为 double 型，因为所有实型数都按 double 型进行计算。这样，1.0/b 的数据类型就是实型（因为 1.0 是实型），而不管 b 为何种类型，结果都为 double 型。

方法 3：利用强制转换，即“f=(double)1/b;”。C 语言允许将一种数据类型强制转换成另一种数据类型，方法是使用“(数据类型）变量”。这样变量即可被转换成括号中的数据类型。

【例 1.8】 字符数据可以字符形式输出，也可以整数形式输出，即输出字符的 ASCII 码值。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    char ch;
    ch='A';
    printf("%c,%d\n",ch,ch);
    return 0;
}
```

运行结果如下：

```
A,65
```

【例 1.9】 字符数据与整数进行算术运算。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    char ch;
    int x;
    ch='A';
    x=ch+32;
    printf("%c,%d\n",x,x);
    return 0;
}
```

运行结果如下：

```
a,97
```

本例说明字符型变量 ch 与整型数 32 相加，结果值为 65＋32=97，因此整型变量 x 获得的值为 97，该数在输出时可以理解为一个整数输出 97，也可理解为一个字符的 ASCII 码值，输出其对应字符“a”。

1.7　数据的输入与输出

输入/输出是对数据的一种重要操作。没有输出的程序是没有用的，没有输入的程序是缺乏灵活性的。程序在多次运行时，用到的数据可能是不同的，因此需要由用户临时输入程序运行所需要的数据。

C 语言本身无输入/输出语句，输入/输出操作均由函数来实现。C 语言提供了多种输入/输出函数，标准 I/O 函数库中有一些公用信息都写在头文件 stdio.h 中，因此，在使用输入/输出函数时，一般应在程序开头添加编译预处理命令：

```
#include <stdio.h>　　或　　#include "stdio.h"
```

在前面的示例程序中，已多次使用格式化输入/输出函数 scanf 和 printf，本节完整地介绍这两个函数。

1.7.1　printf 函数

1．一般形式

printf 函数的一般形式为：

```
printf("格式控制",输出表列);
```

其功能是，按照格式控制部分指定的格式，将输出表列的数据在标准输出设备上输出。

2．格式控制

“格式控制”是用双引号括起来的字符序列，包括格式说明和普通字符两部分，格式说明由“%”和格式字符组成，如%d、%f 等。printf 函数的格式字符及含义见表 1.5。

表 1.5　printf 格式字符

格 式 字 符	含　义
%d（或%i）	输出带符号的十进制整数
%x（或%X）	输出无符号十六进制整数（不输出前导符 0x）
%o	输出无符号八进制整数（不输出前导符 0）
%u	输出无符号十进制整数
%f	以十进制形式输出单、双精度实数（隐含输出 6 位小数）
%e（或%E）	以指数形式输出单、双精度实数（隐含输出 6 位小数）
%g（或%G）	自动选用%f 或%e 格式中输出宽度较短的一种格式输出实数，不输出无意义的 0
%c	输出单个字符
%s	输出字符串
%%	输出%

除了格式控制字符和转义字符，其他字符均是普通字符，普通字符原样输出。下面举例说明格式字符的用法：

```
int a=123,b=-1;
float x=12.34;
char ch=65;
printf("a=%d,b=%d",a,b);          输出结果：a=123,b=-1
printf("a=%X,b=%x",a,b);          输出结果：a=7B,b=ffffffff
printf("a=%o,b=%o",a,b);          输出结果：a=173,b=37777777777
printf("a=%u,b=%u",a,b);          输出结果：a=123,b=4294967295
printf("x=%f",x);                 输出结果：x=12.340000
printf("x=%e",x);                 输出结果：x=1.234000e+001
printf("x=%g",x);                 输出结果：x=12.34
printf("%f%%",1.0/3);             输出结果：0.333333%
printf("ch=%c",ch);               输出结果：ch=A
printf("str=%s","I love China!"); 输出结果：str=I love China!
```

注意：格式字符与其对应的输出项的类型要保持一致。例如，不能用%f 来输出整数。

另外，格式控制字符中，在“%”和其后的格式字符之间，还可以插入附加的格式说明符，如表 1.6 所示。

表 1.6　printf 的附加格式说明符

附加格式说明符	含　义
字母 h	用于输出短整型数据，可加在格式符 d、o、x、u 前面
字母 l	用于输出长整型数据或 double 型数据，可加在格式符 d、o、x、u、f、e（或 E）、g（或 G）前面

续表

附加格式说明符	含　义
m（代表一个正整数）	输出数据的最小宽度，如果数据的实际宽度超过 m，则按实际宽度输出；如果小于 m，则补空格
.n（代表一个正整数）	对于实数，表示输出 n 位小数；对于字符串，表示截取的字符个数
-（负号）	输出的数据或字符在域内向左对齐

下面给出域宽与精度描述符的一些例子。

```
int a=123;
float x=12.345;
printf("a=%5d",a);          输出结果：a=_ _123（_表示空格）
printf("a=%2d",a);          输出结果：a=123
printf("a=%-5X",a);         输出结果：a=7B_ _ _
printf("x=%7.2f",x);        输出结果：x=_ _12.35
printf("x=%.2f",x);         输出结果：x=12.35
printf("x=%10.2e",x);       输出结果：x=_1.23e+001
printf("x=%2E",x);          输出结果：x=1.234500E+001
printf("str=%5.3s","Computer");  输出结果：str=_ _Com
printf("str=%3s","Computer");    输出结果：str=Computer
```

1.7.2　scanf 函数

1．一般形式

scanf 函数的一般形式为：

```
scanf("格式控制",地址表列);
```

其功能是，按照格式控制部分的要求，把从终端输入的数据传送到地址表列指定的内存单元中。

scanf 函数的“格式控制”与 printf 函数的“格式控制”相似。“地址表列”是由若干地址组成的，表示每个输入数据应存储的内存单元地址，可以是变量的地址，也可以是字符串的首地址。在变量名前加符号&表示该变量的地址，如&a。

【例 1.10】 格式化输入与输出。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int a,b,c;
    scanf("%d%d%d",&a,&b,&c);
    printf("a=%d,b=%d,c=%d\n",a,b,c);
    return 0;
}
```

执行 scanf 函数时，如果格式控制部分只包含格式控制字符，不含其他字符，则在输入数据时，数据之间可用一个或多个空格隔开，也可用回车键或跳格键（Tab）隔开。因此，执行上面的程序时，下面的输入均正确：

① 3_ _4_ _ _ _5↙

② 3↙

4_ _ _5↙

③ 3(按 Tab 键)4_ _5↙

输出结果均为：a=3,b=4,c=5

非法的输入：3, 4, 5↙

2．格式说明

scanf 函数的“格式控制”与 printf 函数的“格式控制”类似，以“%”开始，后面跟一个格式字

符，中间也可插入附加格式说明符。表 1.7 列出了 scanf 函数中可以使用的格式字符，表 1.8 列出了 scanf 函数可以使用的附加格式说明符。

表 1.7　scanf 格式字符

格 式 字 符	含　义
%d（或%i）	用来输入十进制整数（正负数均可）
%x（或%X）	用来输入无符号十六进制整数（大小写作用相同）
%o	用来输入无符号八进制整数
%u	用来输入无符号十进制整数
%f, %e, %E, %g, %G	用来输入实数，可以用小数形式或指数形式输入（作用相同）
%c	用来输入单个字符
%s	用来输入字符串，并将字符串送到一个字符数组中。输入时以非空白字符开始，以第一个空白字符结束，并在最后加一个'\0'作为结束标志

说明：

① 可以指定输入数据所占列数，系统自动截取所需数据。

表 1.8　scanf 的附加格式说明符

附加格式说明符	含　义
字母 h	用于输入短整型数据（可用%hd，%ho，%hx）
字母 l	用于输入长整型数据（可用%ld，%lo，%lx）以及 double 型数据（可用%lf 或%le）
m（正整数）	指定输入数据所占宽度（列数）
*	表示本输入项在读入后不赋给相应的变量

例如：

```
int a,b; char c;
scanf("%3d%3c%3d",&a,&c,&b);
```

输入：123ABC456↙后，a=123，b=456，c='A'。

② 如果%后有附加格式说明符“*”，表示跳过它指定的列数。

例如：

```
scanf("%3d%*3c%2d",&a,&b);
```

输入：123456789↙后，a=123，b=78（456 被跳过）。

在利用现有的一批数据时，如果不需要其中某些数据，可用此方法“跳过”它们。

③ 输入实型数据时可以指定数据的宽度，但不能规定数据的精度。

例如：“scanf("%5.2f",&a);”是不合法的。

3．使用 scanf 函数应当注意的问题

① 如果在“格式控制”字符串中除了格式说明以外还有其他字符，则在输入数据时在对应位置应输入这些字符。例如：

scanf("%d,%d,%d",&a,&b,&c); 应当输入：3,4,5↙，不能输入：3 4 5↙。

scanf("%d:%d:%d",&h,&m,&s); 应当输入：12:23:36↙。

scanf("a=%d,b=%d,c=%d",&a,&b,&c); 应当输入：a=1,b=2,c=3↙。

因此，建议在“格式控制”字符串中不要使用其他字符。

② 在用“%c”格式输入字符时，空格字符和转义字符都作为有效字符输入。“%c”只要求读入一个字符，后面不需要用空格作为两个字符的间隔。

scanf("%c%c%c",&c1,&c2,&c3);输入：a b c↙后，c1='a'，c2=' '，c3='b'。

③ 输入数据时，遇到下列情况之一认为该数据输入结束：

- 遇到空格、回车键或跳格键（Tab）。
- 满足指定的宽度。
- 遇到非法的输入。

例如：

```
int a;   float b;   char c;
scanf("%d%c%f",&a,&c,&b);
```

错误输入：1234a123o.26↙后，a=1234，c='a'，b=123.0（而不是希望的 1230.26）。

总之，C 语言的格式输入/输出的规定比较烦琐，重点掌握最常用的一些用法即可，其他部分可在需要时随时查阅。

1.8　C 语言中的宏定义

1.8.1　不带参数的宏定义

1．一般形式

```
#define   标识符   字符串
```

说明：

① #表示这是一条预处理命令。

②“标识符”为所定义的宏名，通常称为符号常量。宏名一般习惯上用大写字母表示，以便与变量名相区别，但也允许用小写字母。

③“字符串”可以是常数、表达式、格式串等。

【例 1.11】 输入半径，计算圆的周长和面积。

参考程序如下：

```
#include <stdio.h>
#define PI 3.14                          // 定义 PI 为一个常数 3.14
int main( )
{
    double r,len,area;                   // 定义 3 个浮点型变量
    printf("Input a radius:");
    scanf("%lf",&r);
    len=2.0*PI*r;
    area=PI*r*r;
    printf("len=%.2f,area=%.2f\n",len,area);
    return 0;
}
```

说明：程序中首先进行宏定义，定义 PI 代表常量 3.14，在 main 函数中计算 len 和 area 时两次使用了 PI，在预处理时经宏展开后 PI 用 3.14 置换，这两个赋值语句变为：

```
len=2.0*3.14*r;
area=3.14*r*r;
```

对于宏定义还需要说明以下几点：

① 宏定义不是语句，在行末不必加分号，如果加上分号则连分号也一起置换。

② 宏定义的作用域为宏定义命令开始到源程序结束。如果要终止其作用域可以使用#undef 命令。

例如：

```
#define PI 3.14
int main( )
{
    …                    PI 在 main 函数中有效
}
#undef PI                终止 PI 的作用域
void fun( )
{
    …                    PI 在 fun 函数中无效
}
```

③ 宏定义是用宏名来表示一个字符串，在宏展开时又以该字符串取代宏名。这只是一种简单的代换，字符串中可以含有任何字符，预处理程序对它不做任何检查。如果有错误，只能在编译已被宏展开后的源程序时才能被发现。

④ 宏名在源程序中若用双引号括起来，则预处理程序不对其进行宏展开。

例如：

```
#define OK 100
int main( )
{
    printf("OK");                  // 不会进行宏展开，程序运行后输出字符串 OK
    printf("%d",OK);               // 宏展开后为“printf("%d",100);”，程序运行后输出 100
    return 0;
}
```

⑤ 宏定义是预处理命令的一个专用名词，与变量定义不同，它不进行内存分配，只是做字符串替换。

例如：

```
#define R 3.0                      // 宏定义，R 不占用内存空间
double r=3.0;                      // 变量定义，r 会占用 8 个字节的内存空间
```

2．宏定义的嵌套

宏定义允许嵌套，即在宏定义的字符串中可以使用已经定义的宏名。在宏展开时由预处理程序层层替换。

【例 1.12】 用宏定义的嵌套形式改写例 1.11。

参考程序如下：

```
#include <stdio.h>
#define PI 3.14
#define R 3.0
#define L 2.0*PI*R                 // PI 和 R 是已定义的宏名
#define S PI*R*R
int main( )
{
    double len,area;
    len=L;                         // 宏展开为 len=2.0*3.14*3.0;
    area=S;                        // 宏展开为 area=3.14*3.0*3.0;
    printf("len=%.2f,area=%.2f\n",len,area);
    return 0;
}
```

1.8.2　带参数的宏定义

1．一般形式

```
#define 宏名(形参表) 字符串
```

【例 1.13】 用带参数的宏定义改写例 1.11。

参考程序如下：

```
#include <stdio.h>
#define PI 3.14
#define L(r) 2.0*PI*r          // r 是形参
#define S(r) PI*r*r
int main( )
{
    double x,len,area;
    scanf("%lf",&x);
    len=L(x);                  // x 是实参，宏展开为 len=2.0*3.14*x;
    area=S(x);                 // x 是实参，宏展开为 area=3.14*x*x;
    printf("len=%.2f,area=%.2f\n",len,area);
    return 0;
}
```

说明：宏定义中的形参只是一个符号代表，它是不需要说明数据类型的。

宏定义也可以用来定义多条语句，在使用宏时，把这些语句替换到源程序内。

【例 1.14】 用多条语句的宏定义改写例 1.11。

参考程序如下：

```
#include <stdio.h>
#define PI 3.14
#define CIRCLE(R,L,S) L= 2.0*PI*R; S=PI*R*R
int main( )
{
    double r,len,area;
    scanf("%lf",&r);
    CIRCLE(r,len,area);            // 宏展开为 len=2.0*3.14*r; area=3.14*r*r;
    printf("len=%.2f,area=%.2f\n",len,area);
    return 0;
}
```

2．使用带参数的宏定义的注意事项

① 带参数的宏定义中，宏名和形参表之间不能出现空格，若有空格则将空格后的所有字符都作为要替换的字符串。例如：

```
#define S (r) PI*r*r                // 这样写认为 S 是宏名，而(r) PI*r*r 是字符串
```

程序中的赋值语句“area=S(x);”将被展开为“area=(r) PI*r*r(x);”，这显然是错误的。

② 带参数的宏定义中，通常将字符串内的形参用小括号括起来，另外还应将整个字符串用小括号括起来，这样才能保证在宏展开时表达式不出现错误。

【例 1.15】 形参和字符串加小括号的情况。

参考程序如下：

```
#include <stdio.h>
#define SQA(x) x*x
#define SQB(x) (x)*(x)
#define SQC(x) ((x)*(x))
int main( )
{
    double n,y1,y2,y3;
    n=3.0;
    y1=SQA(n);                 // 宏展开：y1=n*n; 即 y1=3.0*3.0;
    y2=SQB(n);                 // 宏展开：y2=(n)*(n); 即 y2=(3.0)*(3.0);
    y3=SQC(n);                 // 宏展开：y3=((n)*(n)); 即 y3=((3.0)*(3.0));
    printf("y1=%.2f,y2=%.2f,y3=%.2f\n",y1,y2,y3);
```

```
    n=2.0;
    y1=SQA(n+1);          // 宏展开：y1=n+1*n+1; 即 y1=2.0+1*2.0+1;
    y2=SQB(n+1);          // 宏展开：y2=(n+1)*(n+1); 即 y2=(2.0+1)*(2.0+1);
    y3=SQC(n+1);          // 宏展开：y3=((n+1)*(n+1)); 即 y3=((2.0+1)*(2.0+1));
    printf("y1=%.2f,y2=%.2f,y3=%.2f\n",y1,y2,y3);

    n=1.0;
    y1=2/SQA(n+1);        // 宏展开：y1=2/n+1*n+1; 即 y1=2/1.0+1*1.0+1;
    y2=2/SQB(n+1);        // 宏展开：y2=2/(n+1)*(n+1); 即 y2=2/(1.0+1)*(1.0+1);
    y3=2/SQC(n+1);        // 宏展开：y3=2/((n+1)*(n+1)); 即 y3=2/((1.0+1)*(1.0+1));
    printf("y1=%.2f,y2=%.2f,y3=%.2f\n",y1,y2,y3);
    return 0;
}
```

程序运行结果如下：

```
y1=9.00,y2=9.00,y3=9.00
y1=5.00,y2=9.00,y3=9.00
y1=4.00,y2=2.00,y3=0.50
```

说明：

在例 1.15 中分别定义了 3 个宏，其作用都是求一个数的平方。宏 SQA(x)后的形参和字符串都没加小括号，宏 SQB(x)后的形参 x 加了小括号，而宏 SQC(x)后的形参和字符串都加了小括号。当参数和表达式不同时，它们的计算结果是不同的。

当 n=3.0 时，y1，y2，y3 的结果是一样的，都能正确计算出 3.0 的平方。

当 n=2.0 时，程序的目的是求$(n+1)^2$，但由于宏 SQA(x)的形参 x 没加小括号，实际计算的是 y1=2.0+1*2.0+1，结果是 y1=5.0，出现错误，而 y2 和 y3 的结果是正确的。

当 n=1.0 时，程序的目的是求 $2/(n+1)^2$，使用宏 SQA(x)的结果肯定是错的。虽然宏 SQB(x)的形参 x 加了小括号，但整个字符串没加小括号，所以计算的是 y2=2/(1.0+1)* (1.0+1)，得到 y2=2.0，结果也是错的，而 y3 的结果是正确的。

1.9 算　　法

1.9.1 算法的概念及特性

1．算法

在程序设计中，算法是一系列解决问题的清晰指令，也就是说，能够对一定规范的输入，在有限时间内获得所要求的输出。算法可以理解为由基本运算及规定的运算顺序所构成的、完整的解题步骤，或者按照要求设计好的、有限的、确切的计算序列，并且这样的步骤和序列可以解决一类问题。一般来说，不同的工作，采用的方法和步骤也不同。对于同一个问题可以有不同的解题方法和步骤，也就是有不同的算法。算法有优劣，一般而言，应当选择简单、运算步骤少、运算快、内存开销小的算法。

2．算法的五大特性

① 有穷性：一个算法应当“在合理范围之内”的有限步骤内结束，而不能是无限的；同时应当在执行一定数量的步骤后，算法结束，不能是死循环。

② 确定性：算法中的每一个步骤都不应当产生歧义，其含义应当是确定的。例如，“将成绩优秀的同学名单打印输出”就是有歧义的，“成绩优秀”的含义不明确。

③ 有 0 个或多个输入：所谓输入是指算法执行时从外界获取必要的信息（可以是从键盘输入的数据，也可以是其他部分传递给算法的数据）。一个算法可以没有输入，也可以有输入。

④ 有 1 个或多个输出：即算法必须得到结果，没有结果的算法是没有意义的（结果可以显示在屏幕上，也可以传递给程序的其他部分或文件）。

⑤ 有效性：算法的每个步骤都应当能有效执行，并能得到确定的结果。例如：若 b=0，则 a/b 是不能有效执行的。

1.9.2　算法的表示方法

描述算法有很多种方法，一般有自然语言、流程图、N-S 图、伪代码、程序语言等。最常见的是流程图、N-S 图和伪代码表示等，下面简单介绍。

1．流程图表示算法

流程图用一些框表示各种操作，用箭头表示算法流程。这种用图形表示算法的方法，直观形象，易于理解。

美国标准化协会 ANSI 规定了一些常用的流程图符号，如图 1.5 所示，已被世界各国程序员普遍采用。

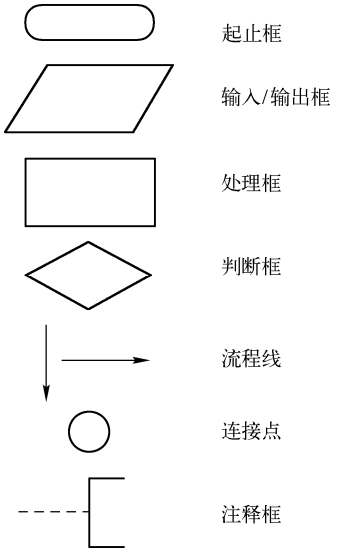

图 1.5　常用流程图符号

- 起止框：表示算法的开始和结束。一般内部只写“开始”或“结束”。
- 输入/输出框：表示算法请求输入需要的数据或算法将某些结果输出。一般内部常填写“输入……”或“打印/显示……”。
- 处理框：表示算法的某个处理步骤，一般内部常填写赋值操作。
- 菱形框（判断框）：对一个给定条件进行判断，并根据给定的条件是否成立来决定如何执行其后的操作。它有一个入口，两个出口。
- 流程线：表示工作流方向，即程序执行的顺序。先执行流程线末端的语句，然后再按照箭头的方向继续执行后面的语句。
- 连接点：用于将画在不同地方的流程线连接起来。同一个编号的点是相互连接在一起的，实际上同一编号的点就是同一个点，只是画不下才分开画的。
- 注释框：注释框不是流程图中必须的部分，不反映流程和操作，它只对流程图中某些框的操作做必要的补充说明，以帮助阅读者更好地理解流程图的作用。

C 语言是一种结构化程序设计语言，结构化程序设计是 20 世纪 60 年代末被提出来的，主要采用自上而下、逐步细化的方法。

结构化程序有 3 种基本结构。

① 顺序结构：最简单的一种基本结构，自顶向下、自左向右顺序地执行程序中的每一条语句。

② 选择结构：也称为分支结构。当判断表达式为真时，执行语句块 A；否则执行语句块 B。

③ 循环结构：分为当型循环和直到型循环。当型循环结构的特点是，判断表达式为真时，执行循环体；表达式为假时退出循环。直到型循环结构的特点是，先执行循环体，直到判断条件表达式为假时，退出循环。

3 种结构的流程图表示如图 1.6 所示。

由上述 3 种结构构成的程序称为结构化程序，也就是说，任何一个结构化程序都可以分解为一个个的基本结构。

用流程图表示的算法直观形象，比较清楚地显示出各个框之间的逻辑关系，因此得到了广泛使用。

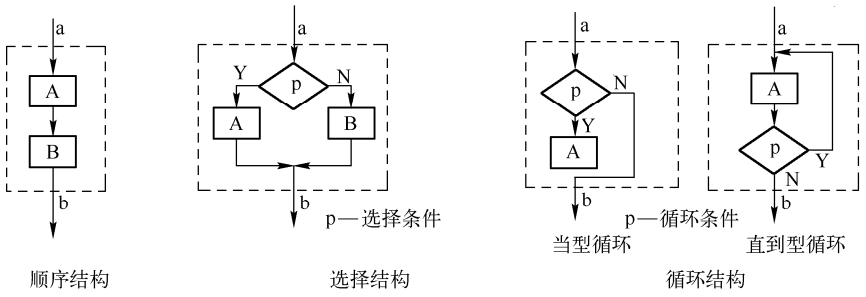

图 1.6　3 种基本程序结构的流程图表示

2．N-S 流程图（盒图）表示算法

1973 年美国学者 I.Nassi 和 B.Shneiderman 提出了一种新的流程图形式，称为 N-S 流程图。N-S 图中，完全去掉了带箭头的流程线，每种结构用一个矩形框表示。

结构化程序设计的 3 种基本结构在 N-S 流程图中的表示方法如图 1.7 所示。

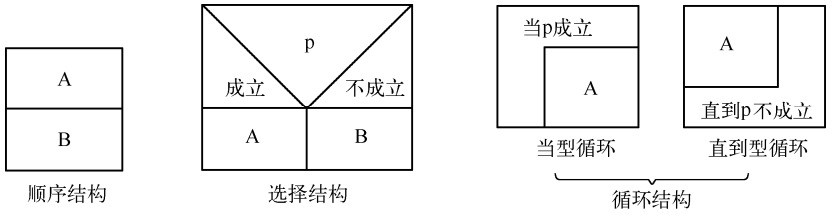

图 1.7　用 N-S 流程图表示的 3 种基本程序结构

N-S 流程图的符号含义说明如下。

（1）顺序结构

先执行 A 操作，再执行 B 操作。

（2）选择结构

当 p 条件成立时，执行 A 操作；当 p 条件不成立时，执行 B 操作。A 操作与 B 操作允许是空操作，即什么都不做。注意，选择结构是一个整体，代表一个基本结构。

（3）循环结构

① 当型循环：当条件 p 成立时，反复执行 A 操作，直到条件 p 不成立为止。当型循环先判断，再决定是否执行循环体，所以在条件 p 一次都不满足时，循环体 A 一次都不执行。

② 直到型循环：直到型循环先执行循环体 A，然后再判断条件 p，直到条件 p 不成立为止。所以循环体至少执行一次。

在一般情况下，循环算法既可以用当型循环，也可以用直到型循环实现。循环结构也是一个整体，同样也代表一个基本结构。

注意：三种结构中的 A、B 框可以是一个简单的操作，也可以是 3 个基本结构之一。也就是说，基本结构可以嵌套。

N-S 流程图表示算法的优点是，N-S 流程图比文字描述更加直观、形象，易于理解；比传统的流程图紧凑易画；废除了流程线，整个算法结构是由各个基本结构按顺序组成的。N-S 流程图的上下顺序就是执行时的顺序，N-S 图表示的算法都是结构化的算法。

3．伪代码表示法

用传统流程图和 N-S 图表示算法，直观易懂，但绘制起来比较麻烦。在设计一个算法时，可能要

反复修改，而修改流程图是件麻烦的事情。因此，流程图适合于表示算法，但不适合于在设计算法过程中使用。为了便于设计算法，常使用伪代码表示算法。

伪代码用介于自然语言和计算机语言之间的文字和符号来描述算法。伪代码不用图形符号，书写方便，格式紧凑，便于向计算机语言程序过渡。

采用介于自然语言和计算机语言之间的文字和符号来描述的算法，是一种非形式化的、比较灵活的混合语言，即相关语句、关键字等采用计算机语言描述，控制过程及限定条件的描述则采用自然语言方式自由地描述。伪代码程序不能在计算机上实际执行，但是严谨的伪代码描述很容易转换为相应的计算机语言程序。

【例 1.16】 猴子吃桃问题：有一堆桃子不知数目，猴子第 1 天吃掉一半，又多吃了一个，第 2 天吃掉剩下桃子的一半又多一个。天天如此，到第 11 天早上，猴子发现只剩一个桃子了，请问这堆桃子原来有多少个？

分析：假设第 1 天有桃子 $peach_1$ 个，第 2 天有 $peach_2$ 个，……，第 9 天有 $peach_9$ 个，第 10 天有 $peach_{10}$ 个，第 11 天有 $peach_{11}$ 个。由于现在只知道第 11 天的桃子数 $peach_{11}$，因此可以借助第 11 天的桃子数 $peach_{11}$ 求得第 10 天的桃子数，计算公式是：$peach_{10}=2\times(peach_{11}+1)$。采用同样的方法，可以由第 10 天推出第 9 天的桃子数：$peach_9=2\times(peach_{10}+1)$，……，由第 2 天推出第 1 天的桃子数，$peach_1=2\times(peach_2+1)$。可以看出 $peach_1$，$peach_2$，$peach_3$，…，$peach_{11}$ 之间存在这样一个关系：

$$peach_i=2\times(peach_{i+1}+1),\quad i=10,9,8,7,\cdots,1$$

这就是本题的数学模型。此外，上述 10 个步骤的计算在形式上是完全一致的，不同的只是变量 peach 的下标而已。因此，可以利用循环的处理方法，并统一采用 peach0 表示前一天的桃子数，用 peach1 表示后一天的桃子数。用伪代码表示算法如下：

```
① peach1=1      // 第 11 天的桃子数，peach1 的初始值
     i=10                    // 计数器的初值为 10
② peach0=2*(peach1+1)        // 计算当天的桃子数
③ peach1=peach0              // 将当天的桃子数作为下一次计算的初值
④ i=i-1                      // 每计算一天，计数器减 1
⑤ 若 i>=1，继续循环执行②；
⑥ 输出 peach0 的值。
```

其中，②～⑤反复地循环执行。

这样的算法已经可以很方便地转化成相应的程序语句了。

伪代码描述算法的优点是：①采用结构化语言提供的结构化控制过程；②可以方便地转换为采用的计算机语言；③易于理解。缺点是：不如图形描述工具形象直观。

1.10　C 语言的产生、发展及特点

1.10.1　C 语言的产生及发展

C 语言是国际上流行的、很有发展前途的计算机高级语言。C 语言适合于作为系统描述语言，既可以用来编写系统软件，也可以用来编写应用软件。

以前操作系统等系统软件主要采用汇编语言来编写。汇编语言依赖于计算机硬件，程序的可读性、可移植性都比较差。为了提高可读性和可移植性，人们希望采用高级语言编写这些软件，但是一般的高级语言难以实现汇编语言的某些操作，特别是针对硬件的一些操作（如内存地址的读/写，位操作等）。人们设法寻找一种既具有一般高级语言特性，又具有低级语言特性的语言，C 语言就在这种情况下应运而生了。

C 语言的产生和发展经历了以下过程：ALGOL 60→CPL→BCPL→B→C→标准 C→ANSI C→ISO C。

① ALGOL 60：一种面向问题的高级语言。ALGOL 60 离硬件较远，不适合编写系统程序。

② CPL（Combined Programming Language，组合编程语言）：一种在 ALGOL 60 基础上更接近硬件的语言。CPL 规模大，实现困难。

③ BCPL（Basic Combined Programming Language，基本的组合编程语言）：对 CPL 进行简化后的一种语言。

④ B 语言：对 BCPL 进一步简化所得到的一种很简单且很接近硬件的语言。B 语言取 BCPL 语言的第一个字母，精练、接近硬件，但过于简单，数据无类型。B 语言诞生后，UNIX 开始用 B 语言改写。

⑤ C 语言：在 B 语言基础上增加数据类型而设计出的一种语言。C 语言取 BCPL 的第二个字母。C 语言诞生后，UNIX 很快用 C 语言改写，并被移植到其他计算机系统中。

⑥ 标准 C、ANSI C、ISO C：C 语言的标准化。

注：最初 UNIX 操作系统是采用汇编语言编写的，B 语言版本的 UNIX 是第一个用高级语言编写的 UNIX 操作系统。在 C 语言诞生后，UNIX 很快用 C 语言改写。C 语言良好的可移植性很快使 UNIX 从 PDP 计算机移植到其他计算机平台，随着 UNIX 的广泛应用，C 语言也得到推广。从此 C 语言和 UNIX 像一对孪生兄弟，在发展中相辅相成，UNIX 操作系统和 C 语言很快风靡全球。

从 C 语言的发展历史可以看出，C 语言是一种既具有一般高级语言特性（ALGOL 60 带来的高级语言特性），又具有低级语言特性（BCPL 带来的接近硬件的低级语言特性）的程序设计语言。C 语言从一开始就被用于编写大型、复杂的系统软件，当然 C 语言也可以用来编写一般的应用程序。也就是说，C 语言是程序员的语言！

IBM PC 微机 DOS、Windows 平台上常见的 C 语言版本如下：

- Borland 公司：Turbo C、Turbo C++、Borland C++、C++ Builder（Windows 版本）。
- Microsoft 公司：Microsoft C、Visual C++（Windows 版本）。

1.10.2　C 语言的特点

C 语言具有以下特点，其中（1）～（6）属于高级语言特性，（7）～（8）属于低级语言特性。

（1）C 语言的语言成分简洁、紧凑、书写形式自由。

（2）C 语言拥有丰富的数据类型。

C 语言具有整型、实型、字符型、数组类型、指针类型、结构体类型、共用体类型等数据类型，能方便地构造更加复杂的数据结构（如使用指针构造链表、栈、树、图等）。

（3）C 语言的运算符丰富、功能更强大。例如：

① C 语言具有复合的赋值运算符“+=、−=、*=、/=、%=”（加、减、乘、除、取余），“>>=、<<=”（右移、左移），“&=、|=、~=”（与、或、非）。例如，x+=5 等价于 x=x+5。

② C 语言有条件运算符“?:”可代替简单的 if-else 语句。

“如果 x 小于或等于 0，y 为 0，否则 y 为 1”可以写成如下条件表达式：

```
y=x<=0?0:1;
```

如果使用 if 语句需要写成如下形式：

```
if (x<=0) y=0;
else y=1;
```

③ 在 C 语言中，赋值操作都被定义为运算符，也就是说，赋值操作本身可以作为表达式的一部分参与运算。例如：

```
while ((ch=getchar( ))!='\n')
    count++;
```

```
ch=getchar( );
while ((ch=getchar( ))!='\n')
    count++;
```

如果改写为一般形式为：

```
ch=getchar( );
while (ch!='\n')
{   count++; ch=getchar( );   }
```

又如，下面的算式是正确的：

```
x=y=z=6;
```

如果改写为一般形式为：

```
z=6; y=6; x=6;
```

（4）C 语言是结构化程序设计语言。

① C 语言具有结构化的控制语句，如 if-else、switch-case、for、while、do-while 等。

② 函数是 C 语言程序的模块单位。

（5）C 语言对语法限制不严格，程序设计灵活。

C 语言不检查数组下标越界，不限制对各种数据的转换（编译系统可能对不合适的转换发出警告，但不限制），不限制指针的使用，而程序的正确性由程序员来保证。

在实践中，C 语言程序编译时会提示：“警告”、“严重错误”。“警告”表示所使用的语法可能有问题，但是有时可以忽略，程序仍然可以完成编译工作，然后执行（但在一般情况下，“警告”往往意味着程序真的有问题，应认真检查）。“严重错误”是不能忽略的，编译系统发现严重错误，就不会产生目标代码。

灵活和安全是一对矛盾，对语法限制的不严格可能也是 C 语言的一个缺点。例如，黑客可能使用越界的数组攻击计算机系统。Java 语言是优秀的网络应用程序开发语言，它必须保证安全性，绝对不允许数组越界。此外，Java 不使用指针，不能直接操作客户计算机上的文件，语法检查相当严格，程序正确性容易保证，但是 Java 在编程时却缺乏灵活性。

（6）C 语言编写的程序具有良好的可移植性。

C 语言编写的程序基本上不需要修改或只需要少量修改就可以移植到其他计算机系统或其他操作系统中。

（7）C 语言可以实现汇编语言的大部分功能：

① 可以直接操作计算机硬件，如寄存器和各种外设 I/O 端口等。

② 可以通过指针直接访问内存物理地址。

③ 类似于汇编语言的位操作可以方便地检查系统硬件的状态。

因此，C 语言适合用于编写系统软件。

（8）C 语言编译后生成的目标代码小、质量高，程序的执行效率高。有资料显示，C 语言只比汇编代码效率低 10%～20%。

习　　题

一、选择题

1．一个 C 语言程序总是从（　　）。

A）主过程开始执行　　B）主函数开始执行

C）子程序开始执行　　D）主程序开始执行

2．C 语言规定：在一个源程序中，main 函数的位置（　　）。

A）必须在最开始　　B）必须在系统调用的库函数的后面

C）可以任意　　D）必须在最后

3．以下叙述中正确的是（　　）。

A）C 语言程序中注释部分可以出现在程序中任意合适的地方

B）花括号“{”和“}”只能作为函数体的定界符

C）构成 C 语言程序的基本单位是函数，所有函数名都可以由用户命名

D）分号是 C 语言语句之间的分隔符，不是语句的一部分

4．C 语言能直接执行的程序是（　　）。

A）源程序　　B）目标程序　　C）汇编程序　　D）可执行程序

5．下面 4 个选项中，（　　）中均是合法的标识符。

A）abc　A_4d　_student　xyz_abc

B）auto　12−a a_b　　ab5.x

C）A_4d　student　xyz_abc　if

D）abc　a_b　union　scan

6．以下选项中不是 C 语言合法常量的是（　　）。

A）'cd'　　B）0.1e+6　　C）"a"　　D）'\011'

7．合法的 C 语言实型常量是（　　）。

A）1.2E0.5　　B）3.14159E　　C）0.5E−3　　D）E15

8．以下叙述不正确的是（　　）。

A）在 C 语言程序中，逗号运算符的优先级最低

B）在 C 语言程序中，APH 和 aph 是两个不同的标识符

C）若 a 和 b 类型相同，则执行语句“a=b;”后，b 的值将放入变量 a 中，而变量 b 的值不变

D）当从键盘输入数据时，对于整型变量只能输入整数，对于实型变量只能输入实数

9．以下能正确地定义整型变量 a，b 和 c 并为它们都赋值为 5 的语句是（　　）。

A）int a=b=c=5;　　B）int a,b,c=5;

C）int a=5,b=5,c=5;　　D）int a=5; int b=c=5;

10．设变量 a 是 int 型，f 是 float 型，i 是 double 型，则表达式“10+'a'+i*f”值的数据类型为（　　）。

A）int　　B）float　　C）double　　D）char

11．表达式 18/4*sqrt(4.0)/8 值的数据类型为（　　）。

A）char　　B）int　　C）float　　D）double

12．若有以下程序段：

```
int c1=1,c2=2,c3;
c3=1.0/c2*c1;
```

则执行后，c3 的值是（　　）。

A）0　　B）0.5　　C）1　　D）2

13．下列选项中，值不等于 0.5 的表达式是（　　）。

A）1.0/2　　B）1/2.0　　C）1/2　　D）(double)1/2

14．若整型变量 a 的原值为 4，则执行表达式“a*=a-=3”后，a 的值是（　　）。

A）1　　B）3　　C）4　　D）13

15．设有定义：“int a; float b;”，执行“scanf("%2d%f",&a,&b);”语句时，若从键盘输入的数据如下：876 543.0↙，则 a 和 b 的值分别是（　　）。

A）876 和 543.000000　　B）87 和 6.000000

C）87 和 543.000000　　D）76 和 543.000000

16．若有输入语句“scanf("%d,%d",&a,&b);”，为使变量 a、b 分别为 5 和 3，则从键盘输入数据的正确形式为（　　）。

A）5 3↙　　B）5,3↙

C）a=5 b=3↙　　D）a=5,b=3↙

17．若有程序段：

```
int a=1234;
double b=123.456;
double c=12345.54321;
printf("%2d,%2.1f,%2.1f\n",a,b,c);
```

其输出结果正确的是（　　）。

A）无输出　　B）12,123.5,12345.5

C）1234,123.5,12345.5　　D）1234,123.4,12345.5

18．下列叙述中正确的是（　　）。

A）预处理命令必须位于源文件的开头

B）在源文件的一行上可以有多条预处理命令

C）宏名必须用大写字母表示

D）宏替换不占用程序的运行时间

19．以下叙述中错误的是（　　）。

A）算法正确的程序最终一定会结束

B）算法正确的程序可以有零个输出

C）算法正确的程序可以有零个输入

D）算法正确的程序对于相同的输入一定有相同的结果

二、填空题

1．C 语言程序的基本单位是_____。

2．一个 C 语言程序如果只有一个函数，则该函数的名字是_____。

3．一个函数包括两部分：_____和_____。

4．若要定义一个符号常量 G 表示地球引力加速度 9.80665，则定义该符号常量的预处理命令为_____。

5．C 语言的源程序必须通过_____和_____后，才能被计算机执行。

6．在 Turbo C 2.0 中，int 类型的变量在内存中占_____个字节的存储空间。在 Visual C++ 6.0 中，int 类型的变量在内存中占_____个字节的存储空间。

7．要求运算对象必须是整数的算术运算符是_____。

8．设 x 和 y 均为 int 型变量，且 x=1，y=2，则表达式“1.0+x/y”的值为_____。

9．C 语言的基本数据类型包括整型、实型、字符型和_____类型。

10．专门用来存放其他变量地址的一种特殊变量称为_____变量。

11．在使用输入输出函数时，需要使用#include 命令包含的头文件是_____（写出文件名）。

12．结构化程序设计所规定的 3 种基本控制结构是_____、_____和

_____。

三、程序填空题

1．若有定义“int a=7,b=9;”，请将下面的输出语句补充完整，使其输出格式为：a=7, b=9。

```
printf("_____",a,b);
```

2．已知字符'B'的 ACSII 码值为 66，以下语句的输出结果是_____。

```
char ch='A';
printf("%c%d\n",ch,ch);
```

3．若 a 是 int 型变量，且 a=5，则下面表达式的值为_____。

```
(a+100)%2+a/2
```

4．若有说明“int i,j,k;”，则表达式“i=10,j=20,k=30,k*=i+j”的值为_____。

5．执行以下程序时输入：1234567↙，则输出结果是_____。

```
#include <stdio.h>
int main( )
{
    int a=1,b;
    scanf("%2d%2d",&a,&b);
    printf("%d %d\n",a,b);
    return 0;
}
```

6．若有程序：

```
#include <stdio.h>
int main( )
{
    int i,j;
    scanf("i=%d,j=%d",&i,&j);
    printf("i=%d,j=%d\n",i,j);
    return 0;
}
```

若要求 i 赋值为 10、j 赋值为 20，应该从键盘输入_____。

7．若有以下宏定义，则输出结果为_____。

```
#define S(x) (x)*(x)
printf("%d\n",1/S(2));
```

四、简答题

1．写出一个 C 语言程序的构成。

2．写出 C 语言程序中函数的结构。

3．主函数 main 在程序中的地位如何，C 语言程序是从哪个函数开始执行的？执行完哪个函数后结束？

4．C 语言中有哪几类语句?

5．开发 C 语言程序的步骤有哪些?

五、编程题

1．请编写一个程序，显示以下 3 行信息。

```
I am a student.
I study in university of Jinan.
I love Jinan.
```

2．若 a=3，b=4，c=5，x=1.2，y=2.4，z=–3.6，u=51274，n=128765，c1='a'，c2='b'。想得到以下输出格式和结果，请写出程序（包括定义变量类型和设计输出）。

a=□3□□b=□4□□c=□5
x=1.200000,y=2.400000,z=–3.600000
x+y=□3.600□□y+z=–1.20□□z+x=–2.40
c1='a'□or□97
c2='b'□or□98
(□表示空格)

3．输入 3 个大写字母，编写程序将其转换成对应的小写字母输出。

4．输入一个三角形的 3 条边长，计算并输出其面积。

假设输入的 3 个边能构成三角形，三角形的面积公式为：

$$\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$$

其中，$s=(a+b+c)/2$。

5．输入一元二次方程 $ax^2+bx+c=0$ 的系数 a、b、c，假设 a≠0 且 $b^2-4ac\geqslant 0$，计算并输出方程的根。

第 2 章　程序基本结构

在上一章我们求解的问题都是一步一步按顺序执行的，这种程序结构称为顺序结构。但在解决实际问题的过程中，对于稍微复杂一些的问题，往往不可能顺序完成。就像经常会遇到岔路口一样，有很多情况需要根据某些条件决定下一步如何操作，还有一些工作需要重复很多次才能完成。这就是本章将要介绍的选择结构和循环结构。

顺序结构、选择结构、循环结构是程序的三种基本结构。已经证明，由三种基本结构组成的算法结构，可以解决任何复杂的问题。由基本结构组成的算法属于结构化算法。

2.1　分 支 结 构

2.1.1　单分支结构

【例 2.1】 输入两个整数，按从小到大的顺序输出这两个数。

分析：对于输入的这两个数 a 和 b，可以比较它们的大小关系，把较小的数存放在第 1 个变量中，把较大的数存放在第 2 个变量中。是否要交换两个变量的值，需要根据 a 和 b 的值来确定，这就产生了分支情况。

C 语言提供了单分支选择结构解决上述问题，形式如下：

```
if (表达式)
    语句;
```

其中，“表达式”是判断条件，只要表达式的值不为 0，就认为条件成立。而“语句”可以是单语句，也可以是复合语句。如果是多条语句，必须使用一对花括号{}构成复合语句。单分支选择结构的执行过程如图 2.1 所示。

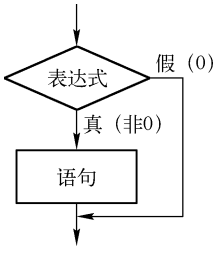

图 2.1　单分支条件语句执行过程

参考程序如下：

```
#include <stdio.h>
int main( )
{
   int a,b,temp;
   printf("Input a,b:");
   scanf("%d%d",&a,&b);
   if (a>b)                          // 判断 a、b 的关系
   { temp=a; a=b; b=temp; }          // a 大于 b，则交换 a，b 的值
   printf("min=%d,max=%d\n",a,b); // 输出最小数 a，最大数 b
   return 0;
}
```

说明：“a>b”是关系运算，如果 a 的值大于 b 的值表示条件成立，表达式的值为真，执行 if 后的语句；否则表达式的值为假，不执行 if 后的语句。

【例 2.2】 输入三个数 x1、x2、x3，按从小到大的顺序输出这三个数。

分析：对于该问题，三个数的大小顺序有多种可能，若依次测试不仅比较复杂，且容易出错。如果能将三个数排序，最小的数保存到 x1，次小的数保存到 x2，最大的数保存到 x3，依次输出 x1、x2、x3 就实现了问题要求。为此，比较 x1 和 x2，如果 x1>x2，则交换两个数，这样 x1 就是两者中较

小的；同理 x1 再与 x3 比较，则 x1 就是三个数中最小的；再比较 x2 和 x3，使 x2<x3。这样经过比较、交换，三个数按从小到大的顺序分别存储在 x1、x2、x3 中。数据的交换在程序设计中经常用到。

算法描述：

① input　x1,x2,x3;

② if x1>x2 then temp=x1,x1=x2,x2=x1　// 交换 x1 和 x2

③ if x1>x3 then temp=x1,x1=x3,x3=x1　// 交换 x1 和 x3

④ if x2>x3 then temp=x2,x2=x3,x3=x2　// 交换 x2 和 x3

⑤ output x1,x2,x3

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int x1,x2,x3,temp;
    printf("Input x1,x2,x3:");
    scanf("%d%d%d",&x1,&x2,&x3);
    if (x1>x2)                          // 交换 x1 和 x2，temp 作为临时变量暂存 x1 的值
    {   temp=x1; x1=x2; x2=temp; }      // 括号部分构成复合语句
    if (x1>x3)
    {   temp=x1; x1=x3; x3=temp; }      // 经过两次交换，x1 为三者中最小的
    if (x2>x3)
    {   temp=x2; x2=x3; x3=temp; }
    printf("%d,%d,%d\n",x1,x2,x3);
    return 0;
}
```

2.1.2　双分支结构

【例 2.3】 输入两个整数，输出其中较大的一个。

分析：输入的是两个整数，需要定义两个整型变量 x 和 y。由于是随机输入的两个数，可以分成 3 种情况：

① 如果 x>y，输出 x;

② 如果 x<y，输出 y;

③ 如果 x=y，输出 x 或者 y 都可以。

输出 x 还是 y，需要根据 x、y 的值来决定，这就产生了分支情况。

C 语言提供了双分支选择结构，形式如下：

```
if (表达式)
    语句 1;
else
    语句 2;
```

表达式的值不为 0，执行“语句 1”，否则执行“语句 2”。同样，“语句 1”和“语句 2”既可以是单语句，也可以是复合语句。双分支选择结构的执行过程如图 2.2 所示。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int x,y;
    printf("Input x,y:");
    scanf("%d%d",&x,&y);
    if (x>=y)                       // 判断 x、y 的关系
       printf("max=%d\n",x);        // x 大于或等于 y，输出 x
```

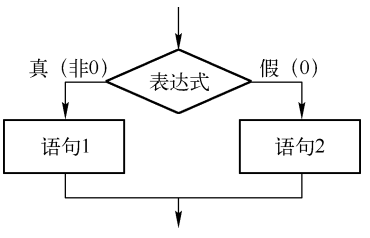

图 2.2　双分支条件语句执行过程

```
    else
       printf("max=%d\n",y);          // x 小于 y，输出 y
    return 0;
}
```

说明：if 语句中的“表达式”可以是任意表达式，一般为关系表达式或逻辑表达式。是执行语句序列 1，还是执行语句序列 2，取决于“表达式”运算的结果。

2.1.3　多分支结构

【例 2.4】 设有分段函数：

$$y=\begin{cases}-e^{2x+1}+3 & (x<-2)\\ 2x-1 & (-2\leqslant x\leqslant 3)\\ 3\lg(3x+5)-11 & (x>3)\end{cases}$$

编写一个程序，输入 x 值，输出 y 值。

分析：本题需要根据判断结果（x 的不同）选择不同的公式来计算 y 值。如果 x 小于−2，y 值为 $-e^{2x+1}+3$；否则，基于 x 不小于−2 的前提，再根据 x 的值是否小于等于 3，分为两种情况。因此，本题需要使用多分支结构。

用嵌套实现的多分支结构的形式为：

```
if (表达式 1)
    语句 1;
else if (表达式 2)
    语句 2;
  …
else if (表达式 n)
    语句 n;
else
    语句 n+1;
```

其功能为按顺序求各表达式的值，如果某一表达式的值为真（非 0），那么执行其后相应的语句，执行完后整个 if 语句结束，其余语句不被执行；如果没有一个表达式的值为真，那么执行最后的 else 语句。执行过程如图 2.3 所示。

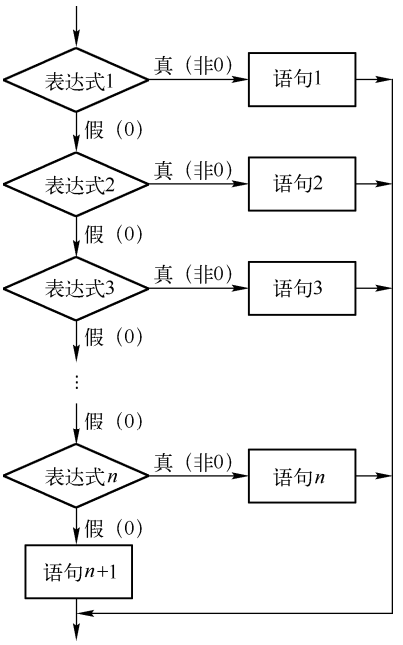

图 2.3　多分支条件语句执行过程

参考程序如下：

```
#include <stdio.h>
#include <math.h>
int main( )
{
    double x,y;
    printf("Input x:");
    scanf("%lf",&x);
    if (x<-2)
        y=-exp(2*x+1)+3;
    else if (x<=3)        // 若没有 else，则运行结果将错误
        y=2*x-1;
    else                  // 本行也可以写成：else if (x>3)
        y=3*log10(3*x+5)-11;
    printf("y=%.2f\n",y);
    return 0;
}
```

说明：在本例中，需要用到两个数学函数，即 exp 和 log10，分别是以 e 为底的指数函数和以 10 为底的对数函数，它们都只有一个参数，计算结果均为 double 型。这两个函数是系统提供的，为了使

用这些数学函数，需要包含头文件“math.h”。即在程序前面增加一行:

```
#include <math.h>
```

其他数学函数也包含在该头文件中。

2.1.4　if 语句的嵌套

所谓 if 语句的嵌套是指 if 语句的 if 分支或 else 分支中，又包含一条或多条 if 语句。一般形式为：

```
if (表达式 1)
    if (表达式 2)  语句 1;
    else  语句 2;
else
    if (表达式 3)  语句 3;
    else  语句 4;
```

对于嵌套结构，应当注意 else 与 if 的配对关系。C 语言规定 else 总是与它前面最近的、而且没有与其他 else 配对的 if 进行配对，特别是 if-else 子句数目不一样时（if 的数量只会大于或等于 else 的数量）。

例如，以下 if 语句中：

```
if (表达式 1)
    if (表达式 2)  语句 1;
    else  语句 2;
```

根据 C 语言规定，else 应与第二个 if 配对。如果希望 else 与第一个 if 配对，可将第二个 if 语句用一对花括号“{ }”括起来，即写成下面的形式：

```
if (表达式 1)
{   if (表达式 2)  语句 1; }
else  语句 2;
```

【例 2.5】 求一元二次方程 $ax^2+bx+c=0$ 的根，a、b 和 c 由键盘输入。

分析：对于一元二次方程有以下几种可能：

① $a=0$，不是一元二次方程；

② $b^2-4ac=0$，有两个相等的实根；

③ $b^2-4ac>0$，有两个不等的实根；

④ $b^2-4ac<0$，有两个共轭复数根。

该问题有多个条件，这就涉及条件语句的嵌套问题。另外，由于涉及开平方运算，有可能出现无限小数。C 语言中，判断一个实数是否等于 0 时，是设定一个较小的值，小于等于该值就认为等于 0。由于使用系统提供的开平方函数 sqrt 和取绝对值函数 fabs，在程序的开始必须用#include 命令包含数学类头文件 math.h。

图 2.4 所示为用 N-S 图描述的求解一元二次方程的算法。

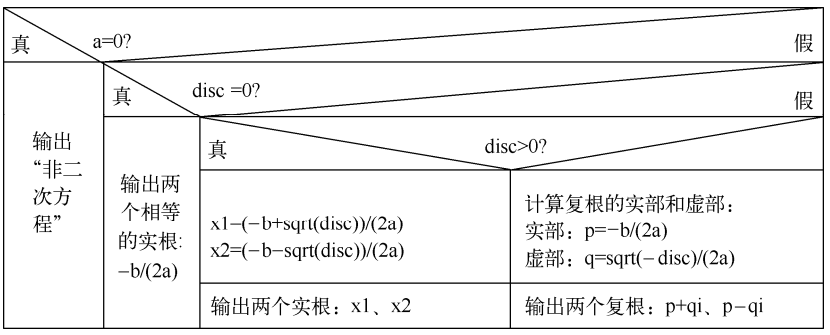

图 2.4　用 N-S 图描述的求解一元二次方程的算法

参考程序如下：

```
#include <stdio.h>
#include <math.h>                          // 数学类函数头文件
int main( )
{
    double a,b,c;
    double disc,x1,x2,rpart,ipart;
    printf("Input a,b,c:");
    scanf("%lf%lf%lf",&a,&b,&c);
    if (fabs(a)<=1E-6)                     // 相当于 a=0，若不是一元二次方程则程序结束
        printf("not a quadratic.\n");
    else                                   // a 不等于 0，则计算方程的根
    {
        disc=b*b-4*a*c;                    // 计算 disc
        if (fabs(disc)<=1E-6)              // disc 等于 0，输出两个相等的实根
            printf("two equal roots:%.2f\n",-b/(2*a));
        else if (disc>1E-6)                // disc 大于 0，输出两个不等的实根
        {
            x1=(-b+sqrt(disc))/(2*a);
            x2=(-b-sqrt(disc))/(2*a);
            printf("two distinct real roots:%.2f,%.2f\n",x1,x2);
        }
        else                               // disc 小于 0，输出两个不等的虚根
        {
            rpart=-b/(2*a);                // 计算复数的实部
            ipart=sqrt(-disc)/(2*a);       // 计算复数的虚部
            printf("two complex roots:%.2f+%.2fi,%.2f-%.2fi\n",
                    rpart,ipart,rpart,ipart);
        }
    }                                      // 第一个 else 结束
    return 0;
}                                          // main 函数结束
```

在本例中，要特别注意“{ }”的使用，左括号“{”一定与最近的右括号“}”结合成一对，而不能交叉。if-else 在逻辑上是一条语句。

运行结果如下：

① Input a,b,c:1 2 1↙
two equal roots:−1.00
② Input a,b,c:1 0 −1↙
two distinct real roots:1.00,−1.00
③ Input a,b,c:2 1 2↙
two complex roots:−0.25+0.97i,−0.25−0.97i

【例 2.6】 某公园门票的票价是每人 10 元，一次购票满 30 张，每张可以少收 1 元。试编写自动计费系统程序。

分析：这个问题并不能简单地根据购票数是否大等于 30 计算相应的费用。因为若购票 29 张，member（购票数）×10（单价）=290（元），若购票 30 张，金额为 270 元，这样显然对购票 29 张的团队是不公平的。需要考虑人数少于 30 人的情况下，票款总额是否大于 270。此外，为了完善功能，程序应该计算实收金额与门票的差额。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int member,sum,money,balance;
    printf("Input the number of entering park:");
```

```
    scanf("%d",&member);
    if (member>=30)
        sum=member*9;
    else if (10*member<270)
        sum=10*member;
    else
    {   sum=270; member=30; }
    printf("cost:%d yuan\n",sum);
    printf("Input Money:");
    scanf("%d",&money);
    balance=money-sum;
    printf("Tickets Money Cost Balance.\n");
    printf("%-7d %-5d %-4d %-7d\n",member,money,sum,balance);
    return 0;
}
```

2.1.5　条件运算符

条件运算符“? :”是三目运算符，其功能相当于简单的 if-else 语句。条件表达式的一般形式为：

表达式 1? 表达式 2: 表达式 3

其功能是，先计算“表达式 1”的值，若为真（非 0）则取“表达式 2”的值为整个条件表达式的值；若为假（0），则取“表达式 3”的值为整个条件表达式的值。执行过程如图 2.5 所示。

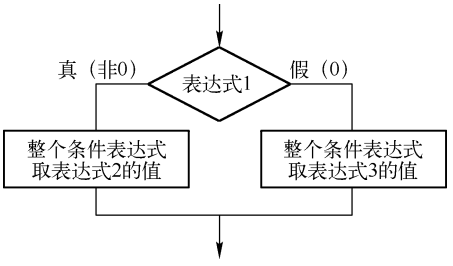

图 2.5　条件运算符执行过程

如果 if 语句中，在表达式为“真”和“假”时，都只执行一个赋值语句，且给同一个变量赋值，则可以使用简单的条件运算符来处理。

例如：

```
max=a>b?a:b
```

该式中，如果 a>b，则 max=a，否则 max=b。将该式用 if 语句表示为：

```
if (a>b) max=a;
else max=b;
```

说明：

① 条件运算符的优先级高于赋值运算符，低于关系运算符和算术运算符。例如：

```
max=a>b?a:b    等价于：    max=((a>b)?a:b)
```

② 条件运算符的结合性为“自右向左”。例如：

```
a>b?a:c>d?c:d   等价于：    a>b?a:(c>d?c:d)
```

③ 表达式 2 和表达式 3 不仅可以是数值表达式，还可以是赋值表达式或函数表达式。例如：

```
a>b?(a=100):(b=100)
a>b?printf("%d\n",a):printf("%d\n",b)
```

④ 表达式 1、表达式 2 和表达式 3 的类型都可以不同。条件表达式值的类型是表达式 2 和表达式 3 中类型较高的类型。例如：

```
x>y?1:1.5          // x>y 时条件表达式的值为 double 型数据 1.0
```

【例 2.7】 输入一个字符，如果是大写字母，则转换为小写字母，如果不是则不转换。输出最后得到的字符。

分析：英文小写字母是大写字母的 ASCII 码值加 32，由于 C 语言允许字符型数据与值在 0～127 范围内的整型数据之间通用，因此小写字母转换为大写字母可用 ch=ch+32 实现。

参考程序如下：

```
#include<stdio.h>
int main( )
{
    char ch;
    scanf("%c",&ch);
    ch=(ch>='A'&&ch<='Z')?(ch+32):ch;
    printf("%c",ch);
    return 0;
}
```

表达式“ch=(ch>='A'&&ch<='Z')?(ch+32):ch;”中的括号可以不要，但有括号看上去程序更加清晰。

2.1.6　switch 语句

【例 2.8】 在学生成绩管理中，成绩经常要在百分制与等级制之间进行转换。90 分以上为 A 等，80～89 分为 B 等，70～79 分为 C 等，60～69 分为 D 等，其余为 E 等。编制程序，根据输入的百分制分数，输出对应的等级。

分析：设用变量 score 表示成绩，而等级是依据 score 的值变化的，共有 5 种情况，如表 2.1 所示。

很明显，这是一个多分支问题，利用 if 语句的嵌套完全可以解决这类问题。但是，如果 if-else 语句嵌套过多，会令人眼花缭乱。幸运的是，C 语言提供了另外一种多分支语句——switch 语句。

switch 的基本格式：

```
switch(表达式)
{
  case  常量表达式 1:  语句组 1; [break;]
  case  常量表达式 2:  语句组 2; [break;]
        …
  case  常量表达式 n:  语句组 n; [break;]
  [default:  语句组 n+1; ]
}
```

表 2.1　成绩等级表

成　　绩	等　　级	判 断 方 法
score>=90	A	score/10=10 或 9
80<=score<90	B	score/10=8
70<=score<80	C	score/10=7
60<=score<70	D	score/10=6
score<60	E	score/10=其他值

说明：

① switch 括号后面的表达式，只能是整型、字符型或枚举型表达式。

② 当表达式的值与某个 case 后面的常量表达式的值相等时，就执行此 case 后面的语句。然后，流程控制转移到下一个 case（包括 default）中的语句继续执行。如果不想继续执行，就需要使用 break 语句使流程跳出 switch 结构，即终止 switch 语句的执行（最后一个分支可以不用 break 语句）。

③ 如果表达式的值与所有常量表达式都不匹配，就执行 default 后面的语句（如果没有 default 就跳出 switch，执行 switch 语句后面的语句）。

④ 各常量表达式的值必须互不相同，否则会出现矛盾。

⑤ case 后面如果有多条语句，也不必用“{ }”括起来。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int score,temp;
    printf("Input score of student:");
    scanf("%d",&score);
    temp=score/10;                    // 整除 10，得到 0-10 之间的整数
    switch(temp)
    {
      case 10:
      case 9: printf("A\n"); break;
```

```
        case 8: printf("B\n"); break;
        case 7: printf("C\n"); break;
        case 6: printf("D\n"); break;
        default: printf("E\n");
    }
    return 0;
}
```

本例中，temp=10 和 temp=9 都输出“A”，共用输出语句“printf("A\n");”，所以“case 10:”后面可以没有语句，且一定不能使用 break 语句。但以后的语句不再共享，必须使用 break 语句终止，否则，程序将继续执行下面的语句。

【例 2.9】 某运输公司对用户按路程计算每公里运费。路程越远，每公里运费越低。运费标准如表 2.2 所示。

分析：观察路程，可以发现折扣的里程是 250 的倍数，将其倍数填入表的右侧。设每公里每吨货物的基本运费为 p，货物质量为 w，距离为 s，折扣为 d，则总运费 f 的计算公式为：f=p*w*s*(1−d)。显然，该问题需要采用分支的方法编写程序。由于 c 都是整数，适合采用 switch 语句。p、w、s 需要用户输入，考虑到程序的通用性，最好定义为实型。注意 break 语句的使用与 c 的值有关。

参考程序如下：

表 2.2　运费标准表

路程 km	折　　扣	c=(int)(s/250)
s＜250	没有折扣	0
250≤s＜500	2%	1
500≤s＜1000	5%	2,3
1000≤s＜2000	8%	4,5,6,7
2000≤s＜3000	10%	8,9,10,11
s≥3000	15%	12,13…

```
#include <stdio.h>
int main( )
{
    int c,s;
    double p,w,d,f;
    printf("Input p,w,s: ");
    scanf("%lf%lf%d",&p,&w,&s);
    c=s/250;
    switch(c)
    {
      case 0:   d=0; break;                  // s<250，没有折扣
      case 1:   d=2; break;                  // 250≤s＜500，折扣 2%
      case 2:
      case 3:   d=5; break;                  // 500≤s＜1000，折扣 5%
      case 4:
      case 5:
      case 6:
      case 7:   d=8; break;                  // 1000≤s＜2000，折扣 8%
      case 8:
      case 9:
      case 10:
      case 11: d=10; break;                  // 2000≤s＜3000，折扣 10%
      default: d=15;                         // s≥3000，折扣 15%
    }
    f=p*w*s*(1−d/100.0);
    printf("f=%f\n",f);
    return 0;
}
```

2.2　关系运算和逻辑运算

前面我们已经使用了简单的关系运算，现在对关系运算和逻辑运算做详细介绍。掌握关系运算和逻辑运算对于正确编写分支程序和循环程序是非常重要的。

2.2.1　关系运算符和关系表达式

关系运算即比较运算，将两个值进行比较，判断是否符合或满足给定的条件。C 语言提供如下 6 个关系运算符：

<、<=、>、>=、==、!=

用关系运算符将两个表达式（算术、关系、逻辑、赋值等表达式）连接起来所构成的表达式，称为关系表达式。关系表达式的值是一个逻辑值，只有两种取值，即真或假。C 语言没有逻辑型数据，关系运算的结果是 int 型，以 1 表示关系表达式成立，代表真，以 0 表示关系表达式不成立，代表假。

关系运算的特点：

① 关系运算符都是双目运算符，并且都是从左向右结合的。

② 关系运算符的优先级比算术运算符低，且它们都比赋值运算符高。在关系运算符内部，>、>=、<、<=运算符的优先级相同，==和!=两种运算符的优先级相同，且低于其他 4 种关系运算符。例如：

```
c>a+b     等价于c>(a+b)     // 关系运算符的优先级低于算术运算符
a>b==c    等价于(a>b)==c    //“>”优先级高于“==”
a==b<c    等价于a==(b<c)    //“<”优先级高于“==”
a=b>c     等价于a=(b>c)     // 关系运算符的优先级高于赋值运算符
```

2.2.2　逻辑运算符和逻辑表达式

在执行分支语句时，经常会遇到需要两个或两个以上条件同时满足或只满足一个条件即可的情况。

在 C 语言中，表达式 x>y>z，并不表示 x>y 且 y>z。那么在 C 语言中如何表示，如何将两个条件合并到一起呢？这就需要使用逻辑运算。

C 语言中的逻辑运算有以下 3 种。

① 逻辑与：两个条件需要同时成立（真）结果才成立（真），否则不成立（假）（相当于而且、并且）。用符号“&&”表示。

② 逻辑或：两个条件只要有一个成立（真）时结果就成立（真），否则不成立（假）（相当于或者）。用符号“||”表示。

③ 逻辑非：条件成立（真），取逻辑非后结果不成立（假）；条件不成立（假），取逻辑非后结果成立（真）。用符号“!”表示。

用逻辑运算符（逻辑与、逻辑或、逻辑非）将关系表达式或逻辑量连接起来就构成了逻辑表达式。

逻辑表达式的值是一个逻辑量（真或假）。C 语言编译系统在给出逻辑运算结果时，以 1 代表真，以 0 代表假，但在判断一个量是否为真时，以 0 代表假，以非 0 代表真（即认为一个非 0 的数值为真）。

在一个逻辑表达式中如果包含多个逻辑运算符，则按照以下的优先顺序执行。

① !(非)→&&(与)→||(或)，“!”为三者中最高的。

② 逻辑运算符中的“&&”和“||”的优先级低于关系运算符，“!”高于算术运算符。例如：

```
a>b&&x>y   等价于    (a>b)&&(x>y)
a==b||x==y 等价于    (a==b)||(x==y)
!a||a>b    等价于    (!a)||(a>b)
```

③ 逻辑运算的真值表如表 2.3 所示。

表 2.3　逻辑运算真值表

a	b	!a	!b	a&&b	a||b
非 0	非 0	0	0	1	1
非 0	0	0	1	0	1
0	非 0	1	0	0	1
0	0	1	1	0	0

在逻辑表达式的求解中，并不是所有逻辑运算符都被执行，只是在必须执行下一个逻辑运算符才能求出表达式的解时，才执行该运算符。例如：

① a&&b&&c

只有 a 为真，才需要判别 b 的值；只有 a、b 都为真，才需要判别 c 的值。只要 a 为假，则整个表

达式已经确定为假，就不必判别 b 和 c 了；如果 a 为真，b 为假，则不必判断 c。

② a||b||c

只要 a 为真，整个表达式已经确定为真，就不必判断 b 和 c 了；只有 a 为假，才判断 b；a、b 都为假，才判断 c。

例如，设 a=1, b=2, c=3, d=4, m=n=1，分析执行下面的表达式后 m 与 n 为何值：

```
(m=a>b)&&(n=c>d)
```

由于“a>b”为假（0），所以赋值后 m=0，赋值表达式“m=a>b”的值也为 0。此时，整个表达式的结果已知为 0，所以不进行表达式“n=c>d”的计算，表达式计算结束后，n=1（未改变）。

【例 2.10】 判断某一年是否为闰年。

分析：所谓闰年，是指符合下面两个条件之一的年份：能被 4 整除，但不能被 100 整除；能被 4 整除，又能被 400 整除。因为能够被 400 整除就一定能被 4 整除所以第二个条件可以简化为能被 400 整除。判断闰年的条件可以用一个逻辑表达式表示：

```
(year%4==0&&year%100!=0)||year%400==0
```

表达式为真，则闰年条件成立，该年份是闰年，否则不是闰年。

请读者自己编写判断闰年的程序。

【例 2.11】 利用逻辑运算改写例 2.4 中的分段函数：

$$y=\begin{cases}-e^{2x+1}+3 & (x<-2)\\ 2x-1 & (-2\leqslant x\leqslant 3)\\ 3\lg(3x+5)-11 & (x>3)\end{cases}$$

编写一个程序，输入 x 值，输出 y 值。

参考程序如下：

```
#include <stdio.h>
#include <math.h>
int main( )
{
    double x,y;
    printf("Input x:");
    scanf("%lf",&x);
    if (x<-2)
       y=-exp(2*x+1)+3;
    if (x>=-2&&x<=3)        // 完整的逻辑表达式，应该用逻辑与“&&”连接
       y=2*x-1;
    if (x>3)
       y=3*log10(3*x+5)-11;
    printf("y=%.2f\n",y);
    return 0;
}
```

2.3 循 环 结 构

2.3.1 概述

前面在例 1.16 中曾经分析过猴子吃桃问题：有一堆桃子不知数目，猴子第 1 天吃掉一半，又多吃了一个，第 2 天吃掉剩下桃子的一半又多一个。天天如此，到第 11 天早上，猴子发现只剩 1 个桃子了，问这堆桃子原来有多少个？

其算法如下：

① peach1=1　　// 第 11 天的桃子数，peach1 的初始值
　　i=10　　// 计数器的初值为 10
② peach0=2*(peach1+1)　　// 计算当天的桃子数
③ peach1=peach0　　// 将当天的桃子数作为下一次计算的初值
④ i=i−1　　// 每计算一天，计数器减 1
⑤ 若 i>=1，继续循环执行②；
⑥ 输出 peach0 的值。

该算法中的②～⑤需要重复执行多次，称其为循环。而控制循环继续还是结束要根据 i 的值是否大于等于 1。

在解决实际问题的过程中，许多问题的求解都归结为重复执行的操作，比如数值计算中的方程迭代求根，非数值计算中的对象遍历。

我们之前编写的程序，经编译连接后都可以生成可执行文件（.exe），但只能执行一次，如需要再次执行，必须再次运行程序。如例 2.7，运行一次只能转换一个字符，如果要转换一批字符，需要多次运行程序。通过循环的方式，即可让程序运行一次就转换多个字符。

重复执行就是循环，循环是计算机特别擅长的工作之一。

循环并不是简单地重复，每次循环，操作的数据（状态、条件）都可能发生变化。如上例中“④ i=i−1”每执行一次，就离循环结束接近一次。

循环的动作是受控制的，比如满足一定条件才继续执行，一直循环到某个条件满足或者规定次数结束等。也就是说，重复工作需要进行控制——循环控制。C 语言提供了 3 种循环控制语句（不考虑 goto 和 if 构成的循环），构成了 3 种基本循环结构：

① while 语句构成的循环结构（当型循环）
② do-while 语句构成的循环结构（直到型循环）
③ for 语句构成的循环结构（当型循环）

2.3.2　当型循环 while

while 循环的一般形式为：

```
while (表达式)
  语句;
```

其中，表达式称为“循环条件”，语句称为“循环体”。while 语句的执行过程如图 2.6 所示。

while 语句的执行过程为：

① 先计算 while 后面表达式的值，如果其值为真（非 0）则执行循环体；

② 执行完循环体后，再次计算 while 后的表达式的值，如果其值为真则继续执行循环体，如果其值为假（0），则退出循环。

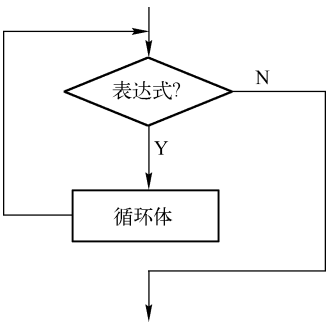

图 2.6　while 循环语句执行过程

使用 while 语句需要注意以下几点。

① while 语句的特点是先计算表达式的值，然后根据表达式的值决定是否执行循环体中的语句。因此，如果表达式的值一开始就为假，那么循环体一次也不执行。

② 当循环体由多个语句组成时，必须用{ }括起来，构成复合语句。

③ 在循环体中应有使循环趋于结束的语句，以避免“死循环”的发生。

【例 2.12】 编写程序计算：1+2+3+…+100。

参考程序如下：

```
#include <stdio.h>
int main( )
```

```
{
    int i=1,sum=0;
    while (i<=100)
    {
       sum=sum+i;
       i++;
    }
    printf("%d\n",sum);
    return 0;
}
```

编写循环程序要注意以下 3 点。

① 遇到数列求和、求积的一类问题，一般可以考虑使用循环解决。

② 注意循环初值的设置。一般对于累加器常设置为 0，累乘器常设置为 1。

③ 循环体中做需要重复的工作，同时要保证使循环逐渐趋于结束。循环的结束由 while 中的表达式（条件）控制。

从这个问题可以看出，循环为编程带来了很大的方便，循环也是程序设计不可或缺的。

再来分析例 2.7（大写字符转换为小写字符），根据我们设计的程序，一次只能转换一个字符，如果要再次转换另一个字符必须重新运行程序，那么是否有办法不重新运行程序而一次转换多个字符呢？实际上这个问题可以方便地借助循环来解决。

【例 2.13】 输入若干字符，如果是大写字母，则转换为小写，如果不是则不转换。输入字符 0 程序结束。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    char ch;
    while ((ch=getchar( ))!='0')      // 如果输入的字符不是 0 则执行循环体
    {
        ch=(ch>='A'&&ch<='Z')?(ch+32):ch;
        printf("%c\n",ch);
    }
    return 0;
}
```

注意：对于该程序，如果输入 0 则直接退出，不会输出。如果现在要求第 1 次输入字符 0 也输出，程序应该如何修改？其实很简单，只要在循环体外，再增加“ch=getchar();”和“printf("%c\n",ch);”语句即可。还有另外一种方法，就是先执行语句，然后再进行条件的判断。实现这种功能的语句就是 do-while 循环。

在这里可使用函数 getchar。getchar 是系统提供的字符输入函数，其作用是从系统隐含指定的输入设备（如键盘）输入一个字符，没有参数。其常用形式如下：

① ch=getchar( );　从键盘输入一个字符，赋值给字符变量 ch。

② getchar( );　作为语句，等待输入，输入值不赋给任何变量。一般用于显示屏幕运行情况，观察运行结果。按任意键程序继续运行。

与其对应的函数 putchar 是字符输出函数。使用格式如下：

```
putchar(c);              // c 为待输出的字符变量
```

例如：

```
c='a';putchar(c);        // 输出字符 a
putchar('\n');           // 输出换行符
```

2.3.3　直到型循环 do-while

do-while 语句的一般形式如下：

```
do
{
    语句;
}while(表达式);
```

其中，表达式称为循环条件，语句称为循环体。

执行过程如下：

① 执行 do 后面的循环体语句。

② 计算 while 后面的表达式的值，如果其值为真（非 0），则继续执行循环体，如果表达式的值为假（0），则退出循环。

do-while 语句的执行过程如图 2.7 所示。

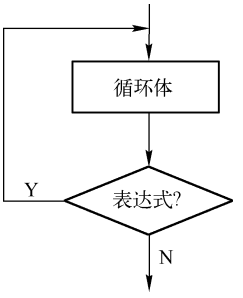

图 2.7　do-while 循环语句执行过程

说明：

① do-while 循环，总是先执行一次循环体，然后再求表达式的值，因此无论表达式是否为真，循环体至少执行一次。

② do-while 循环与 while 循环十分相似，它们的主要区别是：while 循环先判断循环条件再执行循环体，循环体可能一次也不执行；do-while 循环先执行循环体，再判断循环条件，循环体至少执行一次。

③ 当 do-while 语句的循环体语句中有一条语句时，也可以不用花括号，但加上花括号可增加程序的可读性。

④ while 的表达式后面必须有分号“;”。

【例 2.14】 设计一个菜单，格式如下：

```
学生成绩管理系统
1. 学生成绩输入
2. 平均成绩统计
3. 学生成绩查询
4. 学生成绩修改
0. 系统退出
```

要求输入数字 1～4，显示所输的数字；输入数字 0，退出程序；输入其他数字不做任何操作。

分析：根据设计要求，程序应该重复显示，直到输入数字 0，程序退出。这属于直到型循环，因此可以用 do-while 实现。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int x;
    do
    {
        printf("********************\n");
        printf("   学生成绩管理系统   \n");
        printf("********************\n");
        printf("   1. 学生成绩输入   \n");
        printf("   2. 平均成绩统计   \n");
        printf("   3. 学生成绩查询   \n");
        printf("   4. 学生成绩修改   \n");
        printf("   0. 系统退出   \n");
        printf("********************\n");
        scanf("%d",&x);
```

```
        if (x>=1&&x<=4)
            printf("你选择的是第%d 项命令!\n",x);
    }while(x!=0);
    return 0;
}
```

【例 2.15】 计算：1+1/2+1/4+…+1/50

分析：观察数列“1,1/2,1/4,…,1/50”。除第一项为 1 外，其他项分子全部为 1，分母全部是偶数。同样考虑用循环实现。其中，累加器用 sum 表示（初值设置为第一项 1，以后不累加第一项），循环控制用变量 i（i=2～50）控制，数列通项为 1/i。

下面分别是用 while 循环和 do-while 循环实现的参考程序：

```
#include <stdio.h>
int main( )
{
    double sum=1;
    int i=2;
    while (i<=50)
    {
        sum=sum+1.0/i;
        i+=2;
    }
    printf("%f\n",sum);
    return 0;
}
    /* while 方案 */
```

```
#include <stdio.h>
int main( )
{
    double sum=1;
    int i=2;
    do
    {
        sum=sum+1.0/i;
        i+=2;
    }while(i<=50);
    printf("%f\n",sum);
    return 0;
}
    /* do-while 方案 */
```

注意：很多循环都有一个控制循环的变量，如例 2.15 中的变量 i。在进入循环前，为该变量赋初值（表达式 1）；根据该变量的值确定是否结束循环（表达式 2）；在循环体内改变该变量的值（表达式 3）使循环趋于结束。对于此类循环，C 语言提供了一种更紧凑的结构——for 循环。

2.3.4 当型循环 for

for 语句的一般形式如下：

```
for (表达式 1;表达式 2;表达式 3)
    循环体;
```

其中，for 是关键字，其后有 3 个表达式，表达式之间用“;”分隔。3 个表达式可以是任意的表达式，主要用于 for 循环控制。

for 循环执行过程如下：

① 计算表达式 1。

② 计算表达式 2，若其值为真（非 0，表示循环条件成立），则转③；若其值为假（0，表示循环条件不成立），则转⑤。

③ 执行循环体。

④ 计算表达式 3，然后转②。

⑤ 结束循环，执行 for 循环之后的语句。

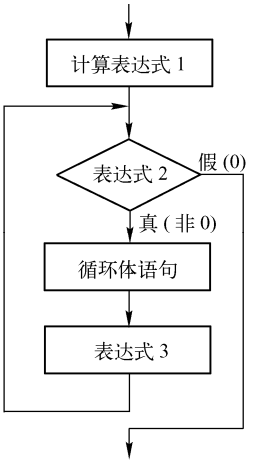

图 2.8　for 循环语句执行过程

for 循环的执行过程如图 2.8 所示。

例 2.15 的 for 循环方案实现的参考程序如下：

```
#include <stdio.h>
int main( )
{
    int i;
    double sum=1;
```

```
    for (i=2;i<=50;i+=2)
       sum=sum+1.0/i;
    printf("%f\n",sum);
    return 0;
}
```

注意：若循环体包含一条以上的语句，则必须用一对花括号括起来构成复合语句，否则将仅把第一条语句作为循环体的内容而产生逻辑错误。

【例 2.16】 求正整数 *n* 的阶乘 *n*!，其中 *n* 由用户输入。

分析：*n*!=1×2×…×*n*。设置变量 fact 为累乘器（被乘数），i 为乘数，兼做循环控制变量。只需让 i 依次从 1 变到 n，同时与 fact 相乘，fact 中结果即为 n!。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    double fact;          // fact 为累乘器，因数值较大，故定义为 double 型
    int i,n;
    scanf("%d",&n);
    for (i=1,fact=1.0;i<=n;i++)
        fact=fact*i;
    printf("%.0f\n",fact);
    return 0;
}
```

可见，for 语句最容易理解、最常用的形式如下：

```
for (循环变量赋初值;循环条件;循环变量修正)
    循环体;
```

说明：

① for 语句中的表达式 1、表达式 2、表达式 3 都可以省略，甚至 3 个表达式都可以同时省略，但是起分隔作用的“;”不能省略。

② 如果省略表达式 1，即不在 for 语句中给循环变量赋初值，则应该在 for 语句前给循环变量赋初值。例如：

```
fact=1.0;                                  fact=1.0; i=1;
for (i=1;i<=n;i++)        等价于：         for ( ;i<=n;i++)
    fact=fact*i;                               fact=fact*i;
```

③ 如果省略表达式 2，即不在表达式 2 的位置判断循环终止条件，也就是认为表达式 2 始终为“真”，此时循环将无终止地进行，则应该在其他位置（如循环体中）安排检测及退出循环的机制。

```
for (i=1,fact=1.0; ;i++)
{
    fact=fact*i;
    if (i==n)
       break;                  // 使用 break 语句跳出循环，结束整个 for 语句的执行
}
```

④ 如果省略表达式 3，即不在此位置进行循环变量的修改，则应该在其他位置（如循环体中）安排使循环趋向于结束的工作。

```
for (i=1,fact=1.0; i<=n; )
{
    fact=fact*i;
    i++;                       // 修改循环变量的值
}
```

⑤ 表达式 1 可以是设置循环变量初值的表达式（常用），也可以是与循环变量无关的其他表达式。表达式 1、表达式 3 可以是简单表达式，也可以是逗号表达式。

```
for (i=1,fact=1.0; ;i++)                    for(i=0,j=100;i<=j;i++,j--)
    fact*=i;                                { temp=a[i]; a[i]=a[j];a[j]=temp; }
```

⑥ 表达式 2 一般为关系表达式或逻辑表达式，也可以是数值表达式或字符表达式。事实上，只要是表达式就可以。

```
for (i=0;(c=getchar( ))!='\n';i+=c) ;        // 此处分号为空语句

for ( ; (c=getchar( ))!='\n';)
    printf("%c",c);
```

小结：C 语言的 for 语句功能强大、使用灵活，可以把循环体和一些与循环控制无关的操作也作为表达式出现，程序短小简洁。但是，如果过分使用这个特点会使 for 语句显得杂乱，降低程序的可读性。建议不要把与循环控制无关的内容放在 for 语句的 3 个表达式中，这是程序设计的良好风格。

从前面循环结构的语法和例子的介绍，可以看出，循环结构由 4 部分组成：

① 循环变量、条件（状态）的初始化。

② 循环变量、条件（状态）检查，以确认是否进行循环。

③ 循环变量、条件（状态）的修改，使循环趋于结束。

④ 循环体处理的其他工作。

2.3.5　几种循环的比较

C 语言中，3 种循环结构（不考虑用 if 和 goto 构成的循环）都可以用来处理同一个问题，但在具体使用时存在一些细微的差别。如果不考虑可读性，一般情况下它们可以相互代替。

① 循环变量初始化：在 while 循环和 do-while 循环中，循环变量初始化应该在 while 和 do-while 语句之前完成；而在 for 循环中，循环变量的初始化可以在表达式 1 中完成。

② 循环条件：while 循环和 do-while 循环只在 while 后面指定循环条件，而 for 循环可以在表达式 2 中指定。

③ 循环变量修改使循环趋向结束：while 循环和 do-while 循环要在循环体内包含使循环趋于结束的操作，而 for 循环可以在表达式 3 中完成。

④ for 语句功能强大。for 循环可以省略循环体，将部分操作放到表达式 2、表达式 3 中。

⑤ while 循环和 for 循环先测试表达式，后执行循环体，而 do-while 循环是先执行循环体，再判断表达式。所以 while 循环和 for 循环是典型的当型循环，而 do-while 循环可以看作是直到型循环。

⑥ 3 种基本循环结构一般可以相互替代，不能说哪种更优越，具体使用哪一种结构依赖于程序的可读性和程序设计者个人程序设计的风格（偏好）。应当尽量选择恰当的循环结构，使程序更加容易理解。尽管 for 循环功能强大，但并不是在任何场合都可以不分条件地使用。

【例 2.17】 1202 年，意大利数学家斐波那契（Fibonacci）出版了他的《计算之书》。在书中提出了一个关于兔子繁殖的问题：如果 1 对兔子每月能生 1 对小兔子（一雄一雌），而每对小兔子在它出生后的第 3 个月里，又能开始生 1 对小兔子，假定在不发生死亡的情况下，由第 1 对出生的小兔子开始，20 个月后会有多少对兔子？

分析：在第 1 个月时，只有 1 对小兔子，过了 1 个月，那对兔子成熟了，在第 3 个月时它们便生下 1 对小兔子，这时有 2 对兔子。第 4 个月，它们再生 1 对小兔子，而另一对小兔子长大，有 3 对兔子。如此推算下去，即可发现一个规律，如表 2.4 所示。

表 2.4　兔子繁殖规律

时间（月）	初生兔子（对）	成熟兔子（对）	兔子总数（对）
1	1	0	1

续表

时间（月）	初生兔子（对）	成熟兔子（对）	兔子总数（对）
2	0	1	1
3	1	1	2
4	1	2	3
5	2	3	5
6	3	5	8
7	5	8	13
8	8	13	21

由此可知，从第 1 个月开始以后每个月的兔子总数是：

1,1,2,3,5,8,13,21,34,55,89,144,233,…

若把上述数列继续写下去，得到的数列被称为斐波那契数列。数列中每个数是前两个数之和，而数列的最初两个数都是 1。

设 f_0=1, f_1=1, f_2=2, f_3=3, f_4=5, f_5=8, f_6=13，则当 n＞1 时，$f_{n+2} = f_{n+1}+f_n$，而 $f_0=f_1=1$。

每个月的兔子总数的参考程序如下：

```
#include <stdio.h>
int main( )
{
    int i,f1=1,f2=1;                    // 第 1、2 个月的兔子数均为 1 对
    int i;
    for (i=1;i<=20;i++)
    {
        printf("%12d %12d",f1,f2);
        if (i%2==0)                     // 控制输出，每行输出 4 个值
           printf("\n");
        f1=f1+f2;                       // 计算第 3 个月的兔子数
        f2=f1+f2;                       // 计算第 4 个月的兔子数
    }
    return 0;
}
```

2.3.6　循环嵌套

一个循环体内又包含另一个完整的循环结构，称为循环嵌套。内层的循环中还可以嵌套循环，形成多层循环。

3 种循环可以相互嵌套。

【例 2.18】 输出九九乘法口诀表。

分析：乘法口诀表的格式是：

1×1=1

1×2=2　2×2=4

……

1×9=9　2×9=18　3×9=27 ……

显然程序应是两层循环，行循环从 1 到 9，用 i 表示；列循环根据行的不同而不同，用 j 表示，循环次数恰好是行数，因此列的循环次数从 1 到 i 即可。“x×y=z”可以通过语句“printf("%d×%d=%-3d",j,i,i*j);”输出。考虑 z 可能是 1 位或 2 位，列之间应该有空格，所以设定宽度为 3。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int i,j;
    for (i=1;i<=9;i++)                          // i 控制一共输出 9 行
    {
        for (j=1;j<=i;j++)                      // 每行输出 j 个表达式
            printf("%d×%d=%-3d",j,i,i*j);       // -3d 表示左对齐，占 3 位
        printf("\n");                           // 输出完一行后换行
    }
    return 0;
}
```

小结：本程序是一个 for 语句的嵌套循环。外层循环控制变量 i 从 1 变到 9，用来表示输出的行数，即从第 1 行到第 9 行；内层循环控制变量 j 从 1 变到 i（i 分别为 1～9），表示每行输出数据的项数（或表达式个数，即外层循环每执行 1 次，内层循环将执行 i 次），每项为 %d*%d=%-3d，如 2*3=6。执行第 1 次外循环时，i=1，j 从 1 变到 1，输出 1 项：1*1=1。执行第 2 次外循环时，i=2，j 从 1 变到 2，输出 2 项：1*2=2　2*2=4，…，依次类推，第 9 行输出 9 项。

2.4　break 语句和 continue 语句

前面介绍的 3 种循环结构都是在执行循环体之前或之后通过对一个表达式的测试来决定是否终止对循环体的执行的。有时出于程序设计的需要，要提前结束循环体。这时可以利用系统提供的 break 语句立即终止循环的执行，而转到循环结构的下一条语句处执行。

循环中还有一种情况是，根据执行结果，本次循环不需要（或不能）执行到最后，就开始下一次循环，这时可利用 continue 语句结束本次循环，开始下一次循环。

2.4.1　break 语句

break 语句的一般形式为：

```
break;
```

break 语句的执行过程是，终止对 switch 语句或循环语句的执行（跳出这两种语句），而转移到其后的语句处执行。

说明：break 语句只用于循环语句或 switch 语句中。在循环语句中，break 常与 if 语句一起使用，表示当条件满足时，立即终止循环。值得注意的是，break 不是跳出 if 语句，而是跳出循环结构。

【例 2.19】 判断 101～200 间有多少个素数，并全部输出。

分析：判断一个数 m 是否素数的常用方法是：用 m 分别去除 2～$\lfloor sqrt(m) \rfloor$，如果能被其中的某个数整除，则表明 m 不是素数；如果所有数都不能整除 m，则 m 是素数。显然，这个问题需要两层循环，外层循环从 101 到 200，分别取出这之间的每个数 m，内层循环用 2～$\lfloor sqrt(m) \rfloor$ 去除 m。如果 m 能被某数 i 除尽，则 m 不是素数，循环可以提前结束。

参考程序如下：

```
#include <stdio.h>
#include <math.h>
int main( )
{
    int m,i,k,count=0,leap;
    for (m=101;m<=200;m+=2)
```

```
    {
        k=(int)sqrt(m);
       leap=1;
        for (i=2;i<=k;i++)
           if (m%i==0)
           { leap=0; break; }                    // 提前结束循环，leap 置 0 表示非素数
        if (leap==1)
        {   printf("%-5d",m); count++; } // 输出 m，素数个数加 1
    }
    printf("The total is %d\n",count);
    return 0;
}
```

循环语句嵌套使用时，break 语句只能跳出其所在的循环，而不能一下子跳出多层循环。要实现跳出多层循环可以设置一个标志变量，控制逐层跳出。例如：

```
…
for (…)
{
  …
  flag=0;                              // 标志变量初始化

  // 如果满足一定条件需要跳出各层循环，先设置标志变量，然后跳出本层循环
  for (…)
  {
    …
    if (…)
    { flag=1; break; }                 // 设置 flag 标记
    …
  }

  if (flag==1) break;                  // 根据标记，继续跳出外层循环
}
…
```

【例 2.20】 从键盘上连续输入字符，并统计其中大写字母的个数，直到输入“换行”字符时结束。

分析：该问题中，while 的条件可以用永真表达式，以 if 语句控制循环的结束。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    char ch;
    int sum=0;
    while (1)
    {
        ch=getchar( );
        if (ch=='\n')
            break;
        if (ch>='A'&&ch<='Z')
            sum++;
    }
    printf("%d\n",sum);
    return 0;
}
```

2.4.2　continue 语句

continue 语句的一般形式为：

```
continue;
```

continue 语句的功能是结束本次循环，即跳过本层循环体中余下尚未执行的语句，接着再一次进行循环条件的判定。

注意：与 break 语句不同，执行 continue 语句并没有使整个循环终止。

在 while 循环和 do-while 循环中，continue 语句使流程直接跳到循环控制条件的测试部分，然后决定循环是否继续执行。在 for 循环中，遇到 continue 后，跳过循环体中余下的语句，对 for 语句中的表达式 3 求值，然后进行表达式 2 的条件测试，最后决定 for 循环是否执行。

【例 2.21】 从键盘输入 30 个字符，统计其中数字字符的个数。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int i,sum=0;
    char ch;
    for (i=1;i<=30;i++)
    {   ch=getchar( );
        if (ch<'0'||ch>'9')
           continue;
        sum++;
    }
    printf("%d\n",sum);
    return 0;
}
```

break 和 continue 对循环的控制如图 2.9 所示。

注意：break 语句和 continue 语句的主要区别是，continue 语句只终止本次循环，而不是终止整个循环结构的执行；break 语句是终止整个循环，不会再进行条件判断。

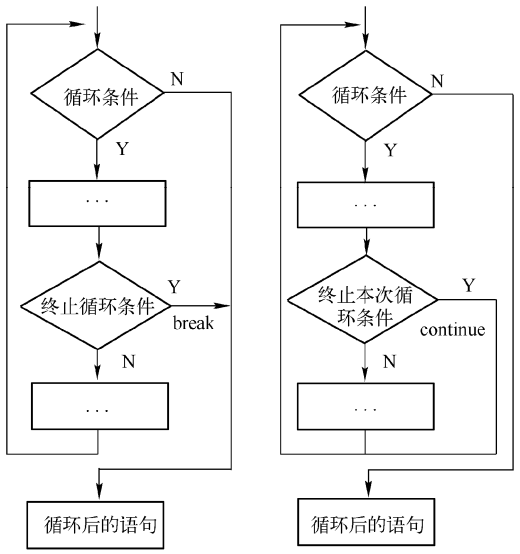

图 2.9　程序流程图比较

2.5　goto 语句

1．语句标号

语句标号的一般形式如下：

```
标识符：
```

它表示程序指令的地址。语句标号的标识符遵守标识符命名规定。

例如：

```
loop:                ERR:
```

2．goto 语句

goto 语句是无条件转移（转向）语句，格式为：

```
goto  语句标号;
```

goto 语句的功能是程序无条件转移到“语句标号”处执行。

结构化程序设计方法主张“限制”（注意不是“禁止”）使用 goto 语句。因为 goto 语句不符合结构化程序设计的准则——模块化。无条件转移使程序结构无规律，可读性变差。但是，任何事情都要一分为二来看，如果能大大提高程序的执行效率，也可以使用。

goto 语句常见的两种用途如下：

① if/goto 构成循环——被 while、do-while 和 for 代替

例 2.22 仅作为 goto 语句标号概念的理解，实际建议不要这样使用。

【例 2.22】 计算 1+2+3…+100。

分析：可用 if 和 goto 构成循环。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int i=1,sum=0;
LOOP:
    sum+=i;
    i++;
    if (i>100) goto PRT;
    goto LOOP;
PRT:
    printf("sum=%d\n",sum);
    return 0;
}
```

② 从循环体跳到循环体外——被 break 和 continue 代替

break 跳出本层循环，continue 结束本次循环。多层循环可以设置一个标志变量，逐层跳出。

2.6　指针程序设计

2.6.1　指针

1．指针的含义

在 C 语言中，变量一旦被定义，系统就为其分配一个确定、唯一的地址。因此，存取该变量的值，可以通过两种方式实现：一种是经常使用的通过变量名存取，另一种就是通过该变量的地址存取。指针变量就是一类存取其他变量或函数的地址的变量，因此可以用指针变量记录某变量的地址，然后通过指针变量存取该变量的值。

需要注意的是，不同类型的变量占用不同数量的内存单元，但是不管变量占用多少个内存单元，它所占用的内存单元一定是连续的。因此，在明确该数据的类型的前提下，只要记住第一个单元的地址（首地址）就可以完整地存取数据。指针变量也有类型，只不过这种类型表示指针变量可以指向的数据的类型。

如图 2.10 所示，指针变量 p 的类型为字符型，值为 0012FF7C，是字符变量 c 的地址。依据这个地址取值，可以得到变量 c 的值。这种存储方式是，以 p 的值为地址，间接地访问变量 c 的值。把通过指针变量访问某内存单元值的方法称为对变量的“间接访问”。相对应的通过变量名访问该内存单元值的方法，称为对变量的“直接访问”。

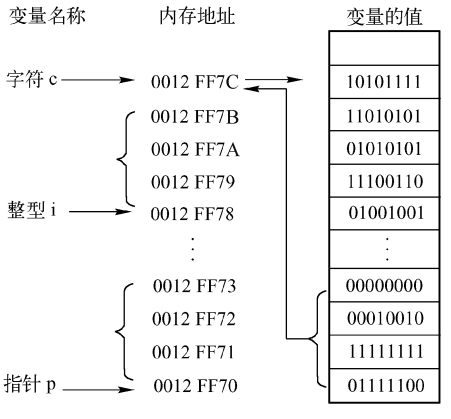

图 2.10　指针变量与变量地址的关系

2．指针变量的定义

在 C 语言中，定义指针变量的一般格式如下：

```
基类型 *指针变量名[=初始值];
```

指针变量名前的*表示该变量是一个指针变量，以示与普通变量的区别。基类型可以是 C 语言中的任何一种数据类

型，是指针变量所指向的对象的类型。初始值可以省略。

例如：

```
int i,*p=&i;
char c,*pc=&c;  // 在定义变量的同时将 c 的地址赋值给 pc
```

利用 p 存取数据，系统就会连续存取 4 个内存单元的数据，而利用 pc 存取数据只存取 1 个单元的数据。由于对 pc 赋予初值&c，指针变量 pc 的值就是变量 c 的地址——通常说，pc 指向了 c。

2.6.2　指针变量的使用

1．指针变量的赋值

指针变量可以通过不同的方法获得一个地址值。

（1）通过地址运算符赋值

地址运算符&是单目运算符，运算对象要放在地址运算符&的右边，用于求出运算对象的地址。通过地址运算符&可以把一个变量的地址赋给指针变量。

例如：

```
float f,*p;
p=&f;
```

以上语句执行后把变量 f 的地址赋值给指针变量 p，指针变量 p 就指向了变量 f。

（2）通过指针变量的初始化赋值

与变量赋初值一样，在定义了一个指针变量之后，其初值也是一个不确定的值，可以在定义变量时给指针变量赋初值。

例如：

```
float f,*p=&f;
```

此语句把变量 f 的地址赋值给指针变量 p，相当于

```
float f,*p;  p=&f;
```

（3）通过其他指针变量赋值

可以通过赋值运算符，把一个指针变量的地址值赋给另一个指针变量，这样两个指针变量均指向同一地址。

例如：

```
int i,*p1=&i,*p2;
p2=p1;
```

以上语句执行后指针变量 p1 与 p2 都指向整型变量 i。

注意：当把一个指针变量的地址值赋给另一个指针变量时，赋值号两边指针变量所指的数据类型必须相同。

例如：

```
int i,*pi=&i;
float *pf;
pf=pi;    // 非法赋值，因为 pf 只能指向 float 型变量，而不能指向整型变量
```

（4）用 NULL 给指针变量赋空值

除了给指针变量赋地址值外，还可以给指针变量赋空值，例如：

```
p=NULL;
```

NULL 是在 stdio.h 头文件中定义的预定义标识符，因此在使用 NULL 时，应该在程序中加上文件包含命令“#include <stdio.h>”。在 stdio.h 头文件中 NULL 被定义成符号常量，与整数 0 对应。

执行以上的赋值语句后，称 p 为空指针。在 C 语言中当指针值为 NULL 时，指针不指向任何有效数据，因此在程序中为了防止错误地使用指针来存取数据，常在指针未使用之前，先赋初值为

NULL。由于 NULL 与整数 0 相对应，所以如下的 3 条语句等价：

```
p=NULL;    p=0;    p='\0';
```

但通常使用“p=NULL;”的形式，因为这条语句的可读性好。NULL 可以赋值给指向任何类型的指针变量。

有关指针变量的赋值，不仅上述 4 种，指针变量还可以指向数组、字符串、结构体、函数、文件及调用标准函数（malloc、calloc）等。

2．指针运算符

通过指针变量可以间接存取变量的数据。为此，C 语言中提供了指针运算符“*”。指针运算符是单目运算符，运算对象只能是指针变量或地址，可以用指针运算符来存取所指向的存储单元中的数据。

若有定义：

```
int x,*p;
p=&x;
```

说明指针 p 指向 x，x 是 p 指向的对象，可以用*p 来引用 x，此时*p 与 x 都代表 x 的值，而 p 与&x 都代表变量 x 的地址。

例如：

```
x=6; 和  *p=6; 等价
scanf("%d",&x);  和  scanf("%d",p); 等价
```

【例 2.23】 由键盘输入一个正整数，求出其最高位数字。请用指针完成本题。

分析：如果输入的数为 1 位数，则该位就是最高位；如果输入为两位数，则除以 10（整除）得到最高位；3 位数应再除以 10；依次类推。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int i,*p;
    p=&i;                          // 让指针变量 p 指向变量 i
    printf("请输入一个正整数：");
    scanf("%d",p);                 // 等价于：scanf("%d",&i);
    while (*p>=10)                 // 当只剩下最高位数字时结束循环
        *p=*p/10;
    printf("最高位数字是：%d\n",*p);
    return 0;
}
```

程序运行结果如下：

```
请输入一个正整数：47586↙
最高位数字是：4
```

【例 2.24】 输入两个数，并按从大到小输出。请用指针完成本题。

分析：对于该问题，可以定义两个指针变量，并让其分别指向不同的变量，通过指针运算实现数据交换。

方法 1 参考程序如下：

```
#include <stdio.h>
int main( )
{
    int x,y,t;
    int *p,*q;
    p=&x;                          // 指针变量 p，指向整型变量 x
    q=&y;                          // 指针变量 q，指向整型变量 y
    scanf("%d,%d",p,q);            // 与 scanf("%d,%d",&x,&y)等价
```

```
        if (*p<*q)                          // 等价于 if (x<y)
        {
            t=*p;                           // 相当于 t=x
            *p=*q;                          // 相当于 x=y
            *q=t;                           // 相当于 y=t
        }
         printf("x=%d y=%d\n",x,y);         // 等价于：printf("x=%d y=%d\n",*p,*q);
    }
```

方法 2 参考程序如下：

```
    #include <stdio.h>
    int main( )
    {
        int x,y;
        int *p,*q,*t;
        p=&x;                               // 指针变量 p，指向整型变量 x
        q=&y;                               // 指针变量 q，指向整型变量 y
        scanf("%d,%d",p,q);                 // 与 scanf("%d,%d",&x,&y)等价
        if (*p<*q)
        {
            // 以下语句仅是指针交换指向，x 和 y 的值并没有交换
            t=p;                            // 指针指向交换
            p=q;
            q=t;
        }
         printf("x=%d y=%d\n",*p,*q);       // 不等价于：printf("x=%d y=%d\n",x,y);
         return 0;
    }
```

指针是 C 语言的重要特色，指针类型是 C 语言的一种特殊数据类型。随着课程的学习，我们会逐渐感受到指针的魅力。正确而灵活地应用指针，可以有效地表示复杂的数据类型、高效地处理字符串和数组，动态地分配内存、处理内存地址，实现主调函数和被调函数之间数据共享等。正确、灵活地使用指针可以设计出结构紧凑、效率更高的应用程序。指针是 C 语言的精华所在，也是 C 语言的难点之一。

2.7　典 型 例 题

1．穷举法求解不定方程

【例 2.25】 百钱百鸡问题：公鸡 5 文钱 1 只，母鸡 3 文钱 1 只，小鸡 1 文钱 3 只。用 100 文钱，买 100 只鸡，问公鸡、母鸡、小鸡各有几只？

分析：该问题有 3 个变量：公鸡数、母鸡数和小鸡数。但只能列出两个方程：

$$\begin{cases} x+y+z=100 \\ 5x+3y+z/3=100 \end{cases}$$

这是一个不定方程，解决这类问题，可以先设 x 的值，再设 y 的值，找出满足条件的 z 值。为找出所有的解答，需要验证所有满足条件的取值。因此，x 应从 1 循环到 20（第二个方程约束条件），在 x 循环的内层 y 从 1 循环到 33，在 y 循环的内层 z 从 3 循环到 100，如果两个方程都满足就是我们需要的解答。

参考程序如下：

```
    #include <stdio.h>
    int main( )
    {
        int x,y,z;
```

```
    printf("***  百钱百鸡问题  ***\n");
    for (x=1;x<=20;x++)
       for (y=1;y<=33;y++)
          for (z=3;z<100;z+=3)                    // 3 的倍数，每次加 3
             if (5*x+3*y+z/3==100 && x+y+z==100)
                printf("cock:%d,hen:%d,chicken:%d\n",x,y,z);
    return 0;
}
```

上面的程序使用了 3 层循环来解决问题，程序结构简单明了，但是设计程序不仅要正确无误，还要注意程序的执行效率。

一般来说，在循环嵌套中，内层循环执行的次数等于该循环嵌套结构中每一层循环重复次数的乘积。

例如，上面的程序中，外层每循环 1 次，第 2 层要循环 32 次，而第 3 层要循环 32×33=1056 次。这样程序执行下来，最内层的 if 语句要执行 19×32×33=20064 次。所以在编写程序时，需要尽可能地减少循环执行的次数，特别是循环嵌套。

对于百鸡问题，由方程组：$\begin{cases} x+y+z=100 \\ 5x+3y+z/3=100 \end{cases}$ 可以导出：$\begin{cases} x=4z/3-100 \\ y=100-x-z \end{cases}$

这样就只有 z 一个未知数，如果知道了 z 就可以求出 x 值，进而求出 y 值。因此，只要将 z 作为循环变量就可以了。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int x,y,z;
    printf("***  百钱百鸡问题  ***\n");
    for (z=3;z<100;z+=3)
    {
        x=4*z/3-100;
        y=100-x-z;
        if (5*x+3*y+z/3==100 && x+y+z==100)          // 满足百钱买百鸡则是一组解，输出
           printf("cock:%d,hen:%d,chicken:%d\n",x,y,z);
    }
    return 0;
}
```

尽管每次循环执行的语句增多了，但循环次数只有 33 次，运行效率大大提高。

从以上分析可以看出，一个好的算法可以提高程序的执行效率，但是要设计出一个好的算法却要花费很大精力，且有时提高效率的同时可能会降低程序的可读性，上例就是如此。

如何掌握好程序的可读性和程序的效率之间的关系，因需要不同，重点也就不同。例如，在处理实时问题时，效率应该优先；而程序量不大，计算机速度又非常快的情况下，效率就不是很重要了。

【例 2.26】 有一本书，被人撕掉了其中一页。已知剩余页码之和为 140，问这本书原来共有多少页？撕掉的是哪一页？

分析：书的页码总是从第 1 页开始，每张纸的页码均为奇数开头，但结束页未必是偶数，一页纸上有两个连续的页码，设为 x、x+1。由前面的分析可知 x 为奇数。设 n 表示原书的页码数，总页码之和为 s，因为剩余页码之和为 140，所以 n<20（$s=\sum_{i=1}^{n} i$）。

由此可以写出不定方程：$s-x-(x+1)=140$，其中 $1 \leqslant x \leqslant n-1$ 且为奇数。

参考程序如下：

```
#include <stdio.h>
int main( )
{
```

```
    int n=1,s=0,x;
    do                                          // 依次测试每一个 n
    {
        s=s+n;                                  // 计算总页码之和
        for (x=1;x<=n-1;x+=2)                   // x 为奇数页
            if (s-x-x-1==140)                   // 页码之和应为 140
                printf("%d,%d,%d,%d",n,s,x,x+1);
        n++;
    }while(n<=20);
    printf("\n");
    return 0;
}
```

这种算法也称为尝试法、枚举法，其核心是全面排查，找出满足条件的所有情况。程序设计简单，但只能解决“有限”的问题。

2．递推法（逆向思维）

【例 2.27】 猴子吃桃问题：猴子第 1 天摘下若干个桃子，当即吃了一半，又多吃 1 个；第 2 天早上又将剩下的桃子吃掉一半，又多 1 个；以后每天早上都吃前一天剩下的一半多 1 个。到第 10 天早上想再吃时，只剩下 1 个桃子了。求第 1 天共摘了多少个桃子。

分析：采取逆向思维的方法，由第 10 天递推到第 9 天，依次向前，一直推到第 1 天，算法分析见例 1.16。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int day,x1,x2;
    day=9;
    x2=1;
    while (day>0)
    {
        x1=(x2+1)*2;        // 第 1 天的桃子数是第 2 天的桃子数加 1 后的 2 倍
        x2=x1;
        day--;
    }
    printf("the total is %d\n",x1);
    return 0;
}
```

3．迭代法解方程

牛顿迭代法的基本思想如下：

设方程 $f(x)=0$ 有实数根，若能够将方程等价地转化成 $x=g(x)$，取一个初始值 x 代入 $x=g(x)$的右端，可得 $x_1=g(x_0)$，依次再计算 $x_2=g(x_1)$，类推可得序列：

$x_{k+1}=g(x_k)$，$k=0,1,2,\cdots$。

称此序列为由迭代函数 $g(x)$产生的迭代序列，x_0 为迭代初始值。若该迭代序列收敛，则它的极限就是方程 $f(x)=0$ 的一个根，x_k 称为方程根的 k 次近似值。称使得迭代法收敛的初始值的取值范围为迭代收敛域。牛顿迭代法如图 2.11 所示。

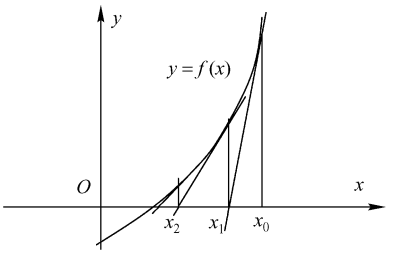

图 2.11　牛顿迭代法示意图

由图 2.11 可以看出，

$$\begin{cases} f'(x_0) = f(x_0)/(x_1 - x_0) \\ x_1 = x_0 - f(x_0)/f'(x_0) \end{cases}$$

这就是牛顿迭代公式。

【例 2.28】 用牛顿迭代法求方程 $2x^3-4x^2+3x-6=0$ 在 1.5 附近的根。

解：$f'(x)=6x^2-8x+3=(6x-8)x+3$

参考程序如下：

```
#include <stdio.h>
#include <math.h>
int main( )
{
    double x,x0,f,f1;
    x=1.5;                  // 假设一个初值
    do
    {
        x0=x;               // 递推，上一个 x 作为下一个 x0
        f=((2*x0-4)*x0+3)*x0-6;
        f1=(6*x0-8)*x0+3;
        x=x0-f/f1;          // 推出新的 x
    }while(fabs(x-x0)>1e-5);
    printf("The root of equation is %5.2f\n",x);
    return 0;
}
```

4．二分法解方程

二分法解方程的基本思想如下：

设根 x 在(a,b)间（注意选取正确的区间），有以下步骤：

步骤①　取 $c=(a+b)/2$，将根区间分为两半，判断根在哪个区间。存在以下 3 种情况；

步骤②　$f(c)\leqslant$精度，则 c 为所求根；

步骤③　若 $f(c)\times f(a)<0$，求根区间在$[a,c]$，令 $b=c$，转①；

步骤④　若 $f(c)\times f(a)>0$，求根区间在$[c,b]$，令 $a=c$，转①。

注意：该算法只能求出一个根，解三次方程适用。

【例 2.29】 求方程 $2x^3-4x^2+3x-6=0$ 在$(-10,10)$之间的根。

参考程序如下：

```
#include <stdio.h>
#include <math.h>
int main( )
{
    double x0,x1,x2,fx0,fx1,fx2;
    do                                  // 限制 x1、x2，必须使 fx1、fx2 异号
    {
        printf("Input x1 & x2:");
        scanf("%lf%lf",&x1,&x2);
        fx1=x1*(x1*(2*x1-4)+3)-6;
        fx2=x2*(x2*(2*x2-4)+3)-6;
    }while(fx1*fx2>0);                  // 若不满足条件则重新输入 x1、x2
    do
    {
        x0=(x1+x2)/2;
        fx0=x0*(x0*(2*x0-4)+3)-6;
        if (fx0*fx1<0)                  // f(c)×f(a)<0
        {
            x2=x0;
            fx2=fx0;
        }
```

```
            else                                    // f(c)×f(a)>=0
            {
                x1=x0;
                fx1=fx0;
            }
        }while(fabs(fx0)>1e-5);                     // 精度不满足要求则继续循环
        printf("x=%6.2f\n",x0);
        return 0;
}
```

5．逻辑问题求解

【例 2.30】 两个乒乓球队进行比赛，各出三人。甲队为 a、b、c 三人，乙队为 x、y、z 三人。已抽签决定比赛名单。有人向队员打听比赛的名单。a 说他不与 x 比，c 说他不与 x 和 z 比。请编程序找出三对赛手的名单。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    char i,j,k;                                     // i 是 a 的对手，j 是 b 的对手，k 是 c 的对手
    for (i='x';i<='z';i++)
        for (j='x';j<='z';j++)
        {
            if (i!=j)
                for (k='x';k<='z';k++)
                {   if (i!=k&&j!=k)
                        if (i!='x'&&k!='x'&&k!='z')
                            printf("order is a--%c\tb--%c\tc--%c\n",i,j,k);
                }
          }
    return 0;
}
```

【例 2.31】 某班有 4 位同学，其中的一位恶作剧，但是谁都不承认。A 说：不是我；B 说：是 C；C 说：是 D；D 说：C 胡说。已知其中 3 个人说的是真话，1 个人说的是假话。编写程序根据这些信息，找出恶作剧的同学。

分析：解这道题需要逻辑思维与判断。下面，我们把 4 个人说的 4 句话写成关系表达式。在声明变量时，让 thisman 表示要找的人，定义它为字符变量：

```
char thisman;
```

令“==”的含义为“是”，“!=”的含义为“不是”

A 说：不是我，写成关系表达式为（thisman!='A'）。

B 说：是 C，写成关系表达式为（thisman=='C'）。

C 说：是 D，写成关系表达式为（thisman=='D'）。

D 说：C 胡说，写成关系表达式为（thisman!='D'）。

如何找到该人？一定是“先假设该人是恶作剧者，然后测试每句话，看有几句是真话”，有 3 句是真话就确定是该人，否则换下一人再试。

比如，先假定是 A 同学，令 thisman='A'，代入到 4 句话中：

A 说：“thisman!='A';”，'A'!='A'为假，值为 0。

B 说：“thisman=='C';”，'A'=='C'为假，值为 0。

C 说：“thisman=='D';”，'A'=='D'为假，值为 0。

D 说：“thisman!='D';”，'A'!='D'为真，值为 1。

显然，不是 A 恶作剧（4 个关系表达式值的和为 1）。

再测试 B 同学，令 thisman='B'，代入到 4 句话中：

A 说：“thisman!='A';”，'B'!='A'为真，值为 1。

B 说：“thisman=='C';”，'B'=='C'为假，值为 0。

C 说：“thisman=='D';”，'B'=='D'为假，值为 0。

D 说：“thisman!='D';”，'B'!='D'为真，值为 1。

显然，不是 B 所为（4 个关系表达式值的和为 2）。

再测试 C 同学，令 thisman='C'，代入到 4 句话中：

A 说：“thisman!='A';”，'C'!='A'为真，值为 1。

B 说：“thisman=='C';”，'C'=='C'为真，值为 1。

C 说：“thisman=='D';”，'C'=='D'为假，值为 0。

D 说：“thisman!='D';”，'C'!='D'为真，值为 1。

显然，就是 C 恶作剧（4 个关系表达式值之和为 3）。

这时，可以理出头绪，即要用所谓枚举法，一个人、一个人地去测试，也就是依次用'A'、'B'、'C'、'D'与'A'、'C'、'D'、'D'比较，4 句话中有 3 句为真，该人即所求。从编写程序的角度看，最好使用循环结构。

在 C 语言中，字符也是有数值的，这个数值就是字符的 ASCII 码值。例如：字符'A'、'B'、'C'、'D'的 ASCII 码值分别为 65、66、67、68。

字符存放在内存中是以 ASCII 码的形式存放的，因此，赋值语句“thisman = 'A';”与“thisman = 65;”是等效的，在内存中存的都是 65。这样，我们就可以用 65、66、67、68 分别代表'A'、'B'、'C'、'D'参与比较了。

以下是主要程序段，请读者自己补充为完整程序：

```
for (k=0;k<=3;k=k+1)
{                                       // 循环体开始/
    thisman=65+k;                       // 产生被试者，依次给 thisman 赋值为'A'、'B'、'C'、'D'
    sum=(thisman!='A')+                 // A 的话是否为真
        (thisman=='C')+                 // B 的话是否为真
        (thisman=='D')+                 // C 的话是否为真
        (thisman!='D');         // D 的话是否为真
    if (sum==3)                         // 若 3 句话为真则找到解
    {
        printf("This man is %c\n",thisman);
        break;                          // 得到答案，结束循环
    }
}
```

习　　题

一、选择题

1．以下是 if 语句的基本形式：

```
if (表达式)
    语句;
```

其中，“表达式”（　　）。

A）必须是逻辑表达式　　B）必须是关系表达式

C）必须是逻辑表达式或关系表达式　　D）可以是任意合法的表达式

2．有数学表达式 x≠y 且 y≥z，则对应的 C 语言表达式为（　　）。

A）(x!=y)||(y>=z)　　B）(x!=y)&&(y>=z)

C）(x<>y)&&(y>=z)　　D）x≠y, y≥z

3．有以下程序段：

```
int a,b,c;
a=10;b=50;c=30;
if (a>b) a=b,b=c; c=a;
printf("a=%db=%dc=%d\n",a,b,c);
```

程序的输出结果是（　　）。

A）a=10b=50c=10　　B）a=10b=50c=30

C）a=10b=30c=10　　D）a=10b=30c=50

4．执行以下程序段的输出结果为（　　）。

```
char c='a';
if ('a'<c<='z')
    printf("LOW");
else
    printf("UP");
```

A）LOW　　B）UP　　C）LOWUP　　D）有语法错误

5．在嵌套使用 if 语句时，C 语言规定 else 总是（　　）。

A）和之前与其具有相同缩进位置的 if 配对

B）和之前与其最近的 if 配对

C）和之前与其最近的且不带 else 的 if 配对

D）和之前的第一个 if 配对

6．有以下计算公式：

$$y=\begin{cases}\sqrt{x} & (x\geqslant 0)\\ \sqrt{-x} & (x<0)\end{cases}$$

若程序前面已在命令行中包含 math.h 文件，不能够正确计算上述公式的程序是（　　）。

A）
```
if (x>=0)
    y=sqrt(x);
else
    y=sqrt(−x);
```

B）
```
y=sqrt(x);
if (x<0)
    y=sqrt(−x);
```

C）
```
if (x>=0)
    y=sqrt(x);
if (x<0)
    y=sqrt(−x);
```

D）
```
y=sqrt(x>=0?x:−x);
```

7．下面的程序段（　　）。

```
int x=0,y=2,z=2;
if (x=y−z)
    printf("****\n");
else
    printf（"####\n");
```

A）有语法错误不能通过编译

B）输出****

C）可以通过编译，但是不能通过连接，因而不能运行

D）输出####

8．以下选项中，当 x 为奇数时，值为 0 的表达式是（　　）。

A）x%2==1　　B）x/2　　C）x%2!=0　　D）x%2==0

9．逻辑运算符两侧的运算对象（　　）。

A）只能是整数 0 或 1　　B）只能是整数 0 或非 0 整数

C）可以是结构体类型的数据　　D）可以是任意合法的表达式

10．设 x、y、t 均为 int 型变量，则执行语句“x=y=3; t=++x||++y;”后，y 的值为（　　）。

A）4　　B）3　　C）1　　D）0

11．有如下程序：

```
#include <stdio.h>
int main( )
{
    int x=1,a=0,b=0;
    switch(x)
    {
        case 0: b++;
        case 1: a++;
        case 2: a++; b++;
    }
    printf("a=%d,b=%d\n",a,b);
    return 0;
}
```

该程序的输出结果是（　　）。

A）a=2,b=1　　B）a=1,b=1　　C）a=1,b=0　　D）a=2,b=2

12．与语句“while(!E);”中的条件“!E”等价的是（　　）。

A）E==0　　B）E!=1　　C）E!=0　　D）~E

13．以下的 for 循环（　　）。

```
for (x=0,y=0; (y!=123)&&(x<4); x++);
```

A）是无限循环　　B）循环次数不定　　C）执行 4 次　　D）执行 3 次

14．执行以下程序的输出结果是（　　）。

```
#include <stdio.h>
int main( )
{
   int x=10,y=10,i;
   for (i=0;x>8;y=++i)
      printf("%d %d",x--,y);
   return 0;
}
```

A）10 1 9 2　　B）9 8 7 6　　C）10 9 9 0　　D）10 10 9 1

15．有以下程序：

```
#include <stdio.h>
int main( )
{
   char k;    int i;
   for (i=l;i<3;i++)
   {
      scanf("%c",&k);
      switch(k)
      {
         case '0': printf ("another ");
         case '1': printf ("number ");
      }
```

```
    }
    return 0;
}
```

程序运行时，从键盘输入：10<回车>，程序执行后的输出结果是（　　）。

A）another number another

B）another number number

C）another number

D）number another number

16．下列叙述中正确的是（　　）。

A）break 语句只能用于 switch 语句中

B）continue 语句的作用是使程序的执行流程跳出包含它的所有循环

C）break 语句只能用在循环体内和 switch 语句内

D）在循环体内使用 break 语句和 continue 语句的作用相同

17．设已有定义“float x;”，则以下对指针变量 p 进行定义且赋初值的语句中正确的是（　　）。

A）float *p=1024;　　B）int *p=(float)x;

C）float p=&x;　　D）float *p=&x;

18．若有说明语句“double *p,a;”，则能通过 scanf 语句正确给输入项读入数据的程序段是（　　）。

A）*p=&a; scanf("%lf",p);　　B）*p=&a; scanf("%f",p);

C）p=&a; scanf("%lf",*p);　　D）p=&a; scanf("%lf",p);

19．已有定义“int k=2; int *ptrl=&k,*ptr2=&k;”，下面不能正确执行的赋值语句是（　　）。

A）k=*ptrl+*ptr2;　　B）ptr2=k;

C）ptrl=ptr2;　　D）k=*ptrl*(*ptr2);

二、填空题

1．若有语句“a=3,b=5;”，则求 a>b 的关系运算结果是_____。

2．在运算符“+、–>、*=、&&”中，其优先级最低的是_____。

3．C 语言提供了 3 种逻辑运算符：&&、||和_____。

4．能表述“20<x<30 或 x<–100”的 C 语言表达式是_____。

5．设 a 是整型变量，命题“a 是偶数”的 C 语言表达式是_____。

6．执行语句“for (i=1;i<5;i++) printf("*");”后，输出结果为_____。

7．C 语言中，break 语句通常用在_____语句和循环语句中。

8．_____变量是一类存取其他变量或函数的地址的变量。

9．若有定义“int i,*p;”，如果需要指针变量 p 指向变量 i，则正确的语句是_____。

10．若有定义“double x;”，请定义指针变量 dp，并使其指向变量 x_____。

三、程序填空题

1．以下程序运行后的输出结果是_____。

```
#include <stdio.h>
int main( )
{
    int a=4,b=3,c=5,t=0;
    if (a>b)
```

```
        { t=a; a=b; b=t; }
        if (a>c) t=a; a=c; c=t;
        printf("%d %d %d\n",a,b,c);
        return 0;
}
```

2．写出以下表达式的值（设 a=1，b=2，c=3，x=4，y=3）。

（1）a+b>c&&b==c_____

（2）b!=c||x+y<=3 _____

（3）a+(b>=x+y)?c−a:y−x _____

（4）!(x=a)&&(y=b)&&0 _____

（5）!(a+b)+c−1&&b+c/2 _____

（6）a||1+'a'&&b&&'c' _____

3．以下程序运行后的输出结果是_____。

```
#include <stdio.h>
int main( )
{
    int i,m=0,n=0,k=0;
    for (i=9;i<=11;i++)
        switch(i/10)
        {
            case 0: m++; n++; break;
            case 10: n++; break;
            default: k++; n++;
        }
    printf("%d %d %d\n",m,n,k);
    return 0;
}
```

4．以下程序运行后的输出结果是_____。

```
#include <stdio.h>
int main( )
{
    int i=1,sum=0;
    while (i<=100)
    {   i++; sum+=i; }
    printf("%d\n",sum);
    return 0;
}
```

5．以下程序运行后的输出结果是_____。

```
#include <stdio.h>
int main( )
{
    int x=2;
    while (x−−);
    printf("%d\n",x);
    return 0;
}
```

6．下面的程序是求 1×2×3×…×10 的值，请填空。

```
#include <stdio.h>
int main( )
{
    int i;
    int s=_____;
    _____
```

```
        s=s*i;
    printf("%d\n",s);
    return 0;
}
```

7．以下程序运行后的输出结果是_____。

```
#include <stdio.h>
int main( )
{
    int n=12345,d;
    while (n!=0)
    {
        d=n%10;
        printf("%d",d);
        n/=10;
    }
    return 0;
}
```

四、编程题

1．输入三个整数 x、y、z，使用条件运算符找出中的最小值并输出。

2．输入一个不多于 4 位的正整数，要求：① 求出它是几位数；② 分别打印出每一位数字；③ 按逆序打印出各位数字。

3．“36 块砖，36 人搬，男搬 4，女搬 3，两个小孩抬一砖，要求一次全搬完。”问男、女、小孩各需搬多少块砖。

4．一个数如果恰好等于它的因子之和，这个数就称为完数。例如，6 的因子为 1、2、3，且 6=1+2+3，因此 6 是完数。编写程序找出 1000 之内的所有完数，并按以下格式输出其因子：

```
6 its factors are: 1 2 3
```

5．一个球从 100 米高度自由下落，每次落地后均反弹为原高度的一半再落下。求它在第 10 次落地时共经过多少米？第 10 次反弹多高？

6．有一分数序列：$\frac{2}{1}, \frac{3}{2}, \frac{5}{3}, \frac{8}{5}, \frac{13}{8}, \frac{21}{13}\cdots$，求出这个序列的前 20 项之和。

7．计算并输出下列多项式的值：

$$F_m = 1+\frac{1}{1!}+\frac{1}{2!}+\cdots+\frac{1}{m!}$$

若输入 m 的值为 5，则输出 F=1.716667。

8．计算并输出 3～m 之间所有素数的平方根之和。若输入 m 的值为 15，则输出为 s=13.536046。

9．计算并输出下列多项式的值：

$$S=\left(1-\frac{1}{2}\right)+\left(\frac{1}{3}-\frac{1}{4}\right)+\cdots+\left(\frac{1}{2n-1}-\frac{1}{2n}\right)$$

10．利用 Maclaurin 公式计算并输出 e^x 的值，精确到 10^{-8}。

$$e^x = 1+x+\frac{x^2}{2!}+\frac{x^3}{3!}+\cdots+\frac{x^n}{n!}$$

若输入 x 的值为 2，则输出为 e^2=7.38905610。

第 3 章　模块化程序设计

通过前两章的学习，我们已经掌握了 C 语言程序设计的 3 种基本结构，能够编写出一些简单的 C 语言程序了，同时也知道函数是 C 语言程序的基本组成单位。

截至目前，我们所编写的程序都只是由一个 main 函数构成的。但随着程序功能的增多、规模的变大，代码量也会越来越多，如果把所有代码都写在 main 函数中，将使整个程序的结构不清晰，阅读不方便，代码重复率高，调试维护困难。这时就应考虑采用模块化程序设计。在 C 语言程序设计中，模块化程序设计是通过函数来实现的。

本章介绍模块化程序设计思想，模块化编程的方法，C 语言的函数的定义及使用，如何利用自定义函数实现模块化编程，变量的作用域、存储类型，指针在函数参数传递中的作用等。

3.1　模块化程序设计思想

模块化是指解决一个复杂问题时自顶向下逐层把系统划分成若干模块的过程。每个模块完成一个特定的子功能，所有模块按某种方法组装起来，成为一个整体，完成整个系统所要求的功能。

以生产台式计算机为例：通常台式机的物理结构由键盘、鼠标、主机、显示器四部分构成，主机又由机箱、电源、硬盘、主板、内存、CPU 及各种板卡等构成。这样一台复杂的计算机可以分解成各个相对独立的子设备分别生产，最后把各部件组合在一起形成一台完整的计算机。按这种划分原则，模块功能相对独立，模块之间联系简单。

在实际应用程序的开发过程中，程序员往往也是把整个程序划分为若干个功能单一、相对独立、较易求解的程序模块，然后分别予以实现，最后再把所有程序模块整合起来。这种在程序设计中逐步分解、分而治之的策略，称为模块化程序设计方法。

函数是 C 语言程序的基本组成单位，因此 C 语言用函数实现程序的模块化。利用函数可以使程序设计变得简单和直观，同时提高程序的易读性和可维护性，而且还可以把程序中经常使用的功能编写成通用函数，以供随时调用，避免代码的重复输入。

【例 3.1】 求表达式 $\frac{K!}{M!+N!}$ 的值。其中，*K*、*M*、*N* 是正整数，由键盘输入。

分析：

① 要想求得该表达式的值，必须先确定 *K*、*M*、*N* 的值，可以通过调用 scanf 输入函数得到，用户要保证输入的三个数是正整数。

② 分别计算 *K*、*M*、*N* 的阶乘。

③ 调用 printf 函数输出 *K*!/(*M*!+*N*!)的结果。

求阶乘的算法前面已经讲过，下面是实现阶乘的参考程序代码段：

```
fn=1;                                   // fn 用于记录阶乘的结果
for (i=1;i<=n;i++)                      // n 是要求阶乘的数
    fn=fn*i;
```

本题参考程序如下：

```
#include <stdio.h>
int main( )
{
```

```
    int     k, m, n;                          // 要计算阶乘的 3 个数
    double fk, fm, fn;                        // 分别存放 3 个数的阶乘结果
    int     i;                                // 循环因子
    scanf("%d%d%d",&k,&m,&n);
    fk=1;                                     // 计算 k 的阶乘
    for (i=1;i<=k;i++)
        fk=fk*i;
    fm=1;                                     // 计算 m 的阶乘
    for (i=1;i<=m;i++)
        fm=fm*i;
    fn=1;                                     // 计算 n 的阶乘
    for (i=1;i<=n;i++)
        fn=fn*i;
    printf("K!/(M!+N!)= %.0f\n",fk/(fm+fn));
    return 0;
}
```

这个程序把所有代码都放在了 main 函数中，这也是我们之前学习编程的主要方式。但仔细观察这个例子可以发现，阶乘的功能相对独立，且只有一个输入值和一个计算结果，并重复使用。如果按照模块化程序设计思想分析，该程序可以划分成如下 3 个模块：输入数据的模块、计算阶乘的模块及输出计算结果的模块。每个模块用一个函数实现，再在主函数 main 中直接调用这些函数即可实现该程序。这样处理，整个程序结构清晰、设计简单直观。由于输入/输出模块系统都提供了相应的函数实现，那么主要任务就是编写自定义函数实现计算阶乘的模块。按照上述分析把例 3.1 改写如下：

```
#include <stdio.h>
double fact(int f)
{
    double   r = 1;
    int      i;
    for (i=1;i<=f;i++)
        r=r*i;
    return r;
}
int main( )
{
    int      k, m, n;
    double   fk, fm, fn;
    scanf("%d%d%d",&k,&m,&n);
    fk=fact(k);                               // 调用阶乘函数，结果赋值给 fk
    fm=fact(m);                               // 调用阶乘函数，结果赋值给 fm
    fn=fact(n);                               // 调用阶乘函数，结果赋值给 fn
    printf("K!/(M!+N!)= %.0f\n",fk/(fm+fn));
    return 0;
}
```

改写后的程序，把计算阶乘的功能独立出来，通过自定义一个 fact 函数实现，这样 main 函数只需调用 3 次 fact 函数即可得到阶乘结果，而无须把每一次阶乘的实现细节都重复列出。

现在 main 函数的职责变成了主要指出“做什么”，而至于“怎么做”则交给其他函数去实现。这样就可以把不需要了解的具体操作细节隐藏起来，使整个程序结构更加清晰，从而降低了修改程序的难度。

由此可以看出，通过模块化程序设计，可使程序各个部分有充分的独立性，设计出的程序结构清晰、简单直观，避免了代码的重复。同时模块化的设计思想还便于团队合作开发，使程序易于调试，大大提高了程序的可读性和可维护性。

3.2　函 数 定 义

根据模块化程序设计的思想，一个较大的程序一般应分为若干个程序模块，每一个模块包括一个或多个函数，每个函数实现一个特定的功能。一个 C 语言程序由一个主函数和若干其他函数构成。程序的执行从主函数的入口开始，到主函数的出口结束，其间主函数可以调用其他函数，其他函数之间也可以相互调用。

在 C 语言中，函数分为以下两种：

① 库函数：由系统提供，用户不用定义，只需用#include 命令包含其头文件，即可直接使用。比如之前经常使用的 printf、scanf、sqrt 等都是 ANSI C 标准定义的库函数。

② 自定义函数：是用户根据具体需求编写的，以完成相应的功能。

迄今为止，前面介绍的 main 函数都是调用标准库函数来完成其功能的。下面重点介绍怎样编写自定义函数。

函数定义就是编写实现函数功能的程序模块。同变量的先定义、后使用一样，函数在使用之前也必须先定义，然后才能调用。函数定义的一般形式为：

```
函数返回值类型 函数名(数据类型 参数 1, 数据类型 参数 2, ……)
{
    函数体
}
```

函数由两部分组成，第一部分称为函数首部（也称函数头），包括函数返回值类型、函数名、参数表；第二部分包含花括号括起来的代码块，称为函数体。

函数总是要实现一定的功能，函数返回值类型就描述了该函数执行完后返回给调用者的结果的数据类型，该返回结果可以是任何有效类型。如果函数执行完不需要返回结果，这个函数的返回值类型应该写成 void。void 是“无类型”的意思。如果什么类型都不指定，则编译器总是假定返回值是 int 类型。

函数名的命名规则和变量命名一样，可以是任何合法的标识符。

函数名后面的参数列表是一个用逗号分隔的变量表，且数据类型和参数必须成对出现。一个函数可以没有参数，这时函数参数列表是空的。但即使没有参数，函数名后面的括号仍然是必须要有的，括号是函数的标志。当然也可以在括号中指定关键字 void，表示没有参数。

函数体必须用一对花括号{ }包围，里面代码的书写规则与 main 函数一样，由声明部分和语句部分组成。

说明：

① 在定义函数时，不要在函数首部的参数列表的右圆括号后使用分号。

② 不要在函数体中再次定义参数列表中出现的变量。

③ 参数列表中的每一个参数都应明确指定一种数据类型。

以 3.1 节计算阶乘的函数为例进行说明：

```
double fact(int f)
{
    double   r = 1;
    int      i;
    for (i=1;i<=f;i++)
        r=r*i;
    return r;
}
```

这是一个求整数阶乘的函数，其中：

① 第 1 行第 1 个关键字 double 表示该函数返回值类型是实型（因为阶乘的结果往往比较大，所以用实型）。

② fact 为函数名，选择有意义的名字可使程序更具可读性。

③ 函数名后面的括号中有一个参数，参数名为 f，其数据类型为整型。函数参数的定义类似于定义变量，先写参数的数据类型，再写参数名。需要注意的是，当参数表中的参数的个数多于 1 个时，它和普通变量的定义稍有区别：

- 多个参数之间用逗号隔开，而不是分号，同时最后一个变量后面不需要符号。例如，普通变量的定义如下：

```
int a;              // 以分号结束
int b;
```

函数中的参数定义为：

```
(int a, int b)      // 以逗号分隔，最后不需要符号
```

- 当两个或多个参数的数据类型相同时，不能同时声明，即数据类型和参数必须成对出现。例如，普通变量定义为：

```
int a,b;            // 多个类型相同的变量可以一起定义
```

函数中的参数定义为：

```
(int a, int b)      // 数据类型和参数必须成对出现。有一个参数，就必须有一个数据类型
```

参数写成(int a, b)的形式是错误的。

④ 一对花括号{ }包围的所有代码，就是函数体。它实现了阶乘的计算，里面的内容和之前学过的 main 函数的书写规则一样。

3.3 函 数 调 用

定义变量是为了使用它。同样，定义函数的目的也是为了使用此函数，而函数的使用是通过函数调用实现的。每一个 C 语言程序都包含一个 main 函数，所有其他函数都直接或间接被 main 函数调用。这与上下级领导关系相似，上级要求下级完成某项任务，并在完成后向上级报告。比如，改写后的例 3.1，main 函数要想得到某个整数的阶乘，可以调用 fact 函数完成这项工作，被调函数 fact 计算了这个数的阶乘，并把结果返回到了主调函数 main 中。主调函数 main 并不知道也不关心被调函数 fact 是怎么样完成指定工作的。

3.3.1 函数调用的形式

函数调用的一般形式为：

```
函数名(实际参数表)
```

函数调用指定了被调用函数的名字和调用函数所需的数据，该数据是通过函数参数提供的。如果被调用函数没有参数，则“实际参数表”可以为空，但括号不能省略。“实际参数表”中的参数可以是常量、变量或表达式，不管是变量还是表达式，有确定的值才有意义。各实参之间用逗号分隔。切记：“实际参数表”的个数、类型和顺序应与被调用函数定义时所要求的参数个数、类型和顺序一致。

根据函数调用在程序中出现的位置，可能有如下 3 种函数调用方式。

（1）函数调用作为独立的语句

如果函数的返回值类型为 void，或者不需要使用其返回值，则可以把函数调用作为一条独立的语句。例如：

```
printf("This is a C program.\n");          // 作为语句，所以结尾必须有分号
```

（2）函数调用出现在表达式中

函数作为表达式的一项出现在表达式中，函数返回值参与表达式的运算。这种方式要求被调用函数必须有返回值。例如：

```
fk=fact(k);                          // 作为赋值表达式的一部分
```

（3）函数调用作为另一个函数的参数

这种方式同样要求被调用函数必须有返回值。例如：

```
printf("k!= %.0f\n",fact(k));        // 作为一个函数的参数使用
```

3.3.2　函数间的参数传递

定义函数的目的是为了实现某个特定功能。当函数被调用时，一般需要给它传递一些数据，供它直接处理或辅助它实现具体的功能。当然有些函数不需要任何外部数据就能完成任务。把需要参数的函数称为有参函数，不需要参数的函数称为无参函数。对于有参函数，定义时函数名后面括号中指定的参数称为形式参数（简称形参）；调用时函数名后面括号中指定的参数称为实际参数（简称实参）。

需要明确的是，函数要不要外部传给它数据、要什么类型的数据、要多少数据、是否有返回值、返回什么类型的值等信息是在定义函数时决定的，而非在被调用时决定。

在调用有参函数时，主调函数要将自己的实参值传递给被调函数的形参。C 语言规定，函数的参数均以“传值调用”方式进行传递，且这种传递是单向的，即只能由实参传给形参，而不能由形参传给实参。

【例 3.2】 输入两个整数，计算其平均值。要求用函数实现平均值的计算。

分析：计算平均值的功能相对独立。题目要求用函数实现平均值的计算，所以在定义函数时，需要明确如下信息：

① 函数是否需要外部传给它数据：该函数的功能是计算任意两个整数的平均值，因此需要外部告之其要计算的两个整数，所以该函数需要外部给其传递数据，是个有参函数。

② 如果是有参函数，那么需要几个参数：经上步分析，应需要两个参数。

③ 如果是有参函数，那么每个参数分别是什么数据类型：根据题目要求，要计算的是整数的平均值，所以两个形参都应是整型。

④ 是否有返回值：根据①的分析，该函数应把计算得到的平均值返回主调函数，所以它应有返回值。

⑤ 如果有返回值，那么返回值的数据类型是什么？两个整数的平均值可能是整数，也可能是小数，所以应定义为实型。

参考程序如下：

```
#include <stdio.h>
double average(int x, int y)                    // 计算平均值函数，有两个形参
{
    double result;                              // result 用于保存计算结果
    result=(x+y)/2.0;                           // 计算平均值，把结果赋给 result
    return result;                              // 把结果返回主调函数
}
int main( )
{
    int a, b;
    double ave;
    scanf("%d%d",&a,&b);                        // 输入要计算平均值的两个整数
    ave=average(a,b);                           // 调用 average 函数，有两个实参
    printf("Average of %d and %d is %f\n",a,b,ave);
    return 0;
}
```

主函数 main 中包含了一个函数调用 average(a,b)。avergae 后面括号内的 a 和 b 是实参，是在主调函数 main 中定义的变量。当发生调用时，主调函数 main 把 a 和 b 的值分别赋值给被调函数 average 中定义的形参 x 和 y，这个过程就是参数传递。

程序的具体执行过程如下：

① 执行主函数中的 scanf 函数，从键盘输入两个数，分别存放到变量 a 和 b 中。

② 调用 average(a, b)。

③ 流程转去执行 average 函数中的语句。此时 average 函数中的形参 x 和 y 就有了明确的值（x 为 a 的值，y 为 b 的值）。通过计算，result 得到了 x 和 y 的平均值，并通过 return 语句把该值返回主调函数中，average 函数的运行至此结束。

④ 流程转回主函数 main 中，从调用 average 函数的地方继续运行，即把 average 函数执行完返回的值赋给变量 ave。

⑤ 执行 printf 函数把结果输出，程序运行结束。

上述调用过程可以用图 3.1 说明。

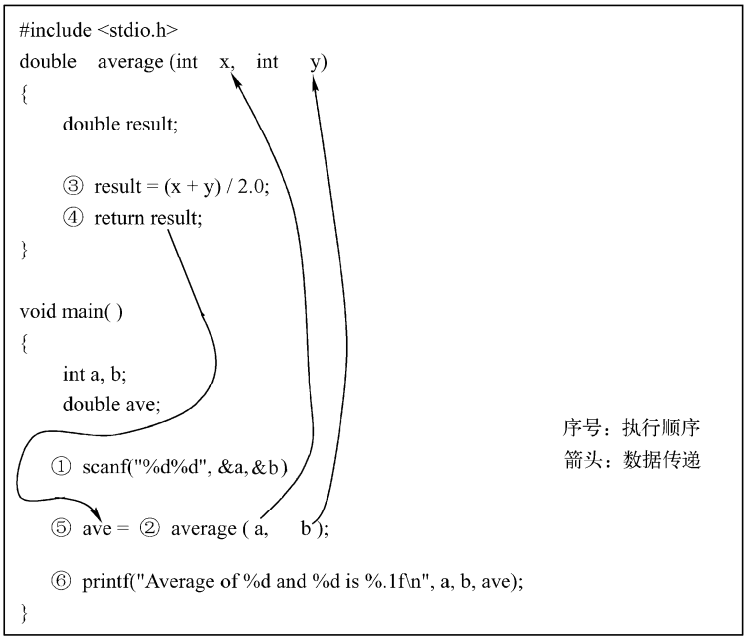

图 3.1　例 3.2 函数调用关系图

说明：

① 实参可以是常量、变量、表达式、函数等。无论实参是何种类型，在进行函数调用时，它们都必须具有确定的值，以便把这些值传递给形参。因此，应预先用赋值、输入等方法，使实参获得确定的值。

② 形参变量只在函数被调用时，才分配内存单元。调用结束时，即刻释放所分配的内存单元。因此，形参只在该函数内有效。调用结束，返回调用函数后，则不能再使用该形参变量。

③ 实参对形参的数据传递是单向的，即只能把实参的值传递给形参，而不能把形参的值反向地传递给实参。

④ 实参和形参占用不同的内存单元，即使同名也互不影响。

3.3.3　函数的返回值

函数的返回值是指函数被调用之后，执行函数体中的程序段所取得的、并返回给主调函数的值。该值可以存储在变量中，也可以进行测试、输出或用于其他用途。例如，调用系统提供的开方函数 sqrt 取得开方值，调用修改后例 3.1 的 fact 函数取得整数阶乘，调用例 3.2 的 average 函数得到两个整数的平均值等。

函数的返回值只能通过 return 语句返回到主调函数。return 语句的一般形式为：

```
return 表达式;    或    return;
```

其中，表达式可以加括号：

```
return(表达式);
```

return 语句的功能是立即结束当前函数的执行，并返回到主调函数中。

如果 return 后面有表达式，那么该表达式会被计算，并作为被调函数的返回值传送给主调函数；对于返回值类型为 void 的函数，return 后面不需要任何表达式。事实上，这样的函数也可以不用 return 语句，程序执行到函数结束的右花括号时，此函数的执行就会结束，程序跳转回主调函数。

说明：

① 有返回值的函数必须有 return 语句。

② 在函数中允许有多个 return 语句，但每次调用只能有一个 return 语句被执行，因此函数的返回值只能有一个。

③ 函数返回值的数据类型可以是除数组和共用体类型（第 4、5 章介绍）以外的任意类型。

④ return 语句中表达式的类型和函数定义中的函数返回值类型应保持一致。如果两者不一致，则以函数返回值类型为准。对数值型数据，系统会自动进行类型转换。最终的返回值类型由函数定义时的函数返回值类型决定。

3.4　函数的原型与声明

C 语言中，形参和实参的个数、类型和顺序是确定的，为保证其一致性，在函数调用之前应对所调用的函数进行声明，指出该函数的返回值的类型及形参的个数和类型，编译器根据此信息对调用的函数进行语法检查，保证参数及返回值的正确性。

函数声明的一般形式：

```
函数返回值类型 函数名(形参类型 1 形参名 1, 形参类型 2 形参名 2, ……);
```

说明：

① 在函数声明中，由于编译系统不检查参数名，因此形参表中可以只写形参的数据类型，而不写形参名，从而可以简化为以下形式：

```
函数返回值类型 函数名(形参类型 1, 形参类型 2, ……);
```

② 函数声明中，函数返回值类型、函数名、参数个数、参数类型和参数顺序应与函数定义保持一致。

③ 函数声明的位置：一种是在主调函数中对被调函数进行声明；另一种是在所有函数的外部进行函数声明（推荐使用）。

【例 3.3】 输入两个整数，计算其平均值。要求用函数实现平均值的计算，且该函数的定义出现在主函数的后面。

参考程序如下：

```
#include <stdio.h>
int main( )
{
```

```
    int a, b;
    double ave;
    double average(int x, int y);              // 声明 average 函数
    scanf("%d%d",&a,&b);
    ave=average(a,b);                          // 对 average 函数进行调用
    printf("Average of %d and %d is %f\n",a,b,ave);
    return 0;
}
// 计算平均值
double average(int x,int y)                    // 该函数定义出现在主调函数 main 的后面
{
    double result;
    result=(x+y)/2.0;
    return result;
}
```

函数的外部声明举例：

```
#include <stdio.h>
// 以下 3 行是函数原型的声明，它们出现在所有函数之前，且在函数外部
double test1(int a, int b);
int test2(char c, int d);
char test3(double e, float f);
int main( )
{
    // 在 main 函数中要调用 test1，test2，test3 函数，不需再对这 3 个函数进行声明
    …
}
// 下面定义被 main( )函数调用的 3 个函数
double test1(int a, int b)                     // 定义 test1 函数
{
    …
}
int test2(char c, int d)                       // 定义 test2 函数
{
    …
}
char test3(double e, float f)                  // 定义 test3 函数
{
    …
}
```

如果函数在调用语句前已经定义，则可以不声明，但为提高程序的可读性，建议对自定义函数都要进行声明。

对函数的定义和声明不是一回事。定义是指对函数功能的确立，包括指定函数名、函数值类型、形参类型、函数体等，它是一个完整、独立的函数单位。而声明的作用则是把函数的名字、函数类型，以及形参类型、个数和顺序通知编译系统，以便在调用该函数时系统按此进行对照检查（例如函数名是否正确，实参与形参的类型和个数是否一致）。从程序中可以看到对函数的声明与函数定义中的函数首部基本上是相同的。因此，可以简单地照写已定义的函数的首部，再加一个分号，就成为了对函数的声明。在函数声明中也可以不写形参名，而只写形参的类型。

另外，当调用系统函数时，使用头文件实现对函数原型的声明。通过#include 命令包含这些头文件，就可以让编译器找到该函数。例如，在程序中调用 printf、scanf 等系统函数，使用“#include <stdio.h>”方式，达到声明的效果。也可以把自己所定义的函数原型放在头文件中，只要在程序中包含此头文件，就可以使用这些函数。

3.5　函数的嵌套与递归

3.5.1　函数的嵌套调用

C 语言的函数定义都是互相平行、独立的，不存在隶属关系。也就是说，在定义函数时，一个函数内部不能再定义另一个函数，即不能嵌套定义函数，但可以嵌套调用函数。所谓嵌套调用是指在一个函数的定义中出现对另一个函数的调用，即在被调函数中又调用其他函数。

【例 3.4】 编程求 $\sum_{x=1}^{n} x^k$ ，输入 k 和 n 的值。其中，k 和 n 是正整数。

分析：假设输入 k=3, n=5，该题就变成求 $1^3+2^3+3^3+4^3+5^3$ 的值。从总体上看，该程序就是一个求和运算，所以，按照模块化程序设计思想，该问题可以分解为数据输入、求和、数据输出 3 个模块。而求和模块中参与求和的数又是一个乘方运算，所以求和部分又可分解出求乘方的模块。这样，该问题实际被分为 4 个模块：

① 输入 n 和 k 的值；

② 乘方运算，即计算 x^k；

③ 求和运算，即计算 $1^k+2^k+\cdots+n^k$；

④ 输出结果。

①和④可以直接调用系统函数实现。②和③需要编写自定义函数实现：一个用来计算乘方的 power 函数、一个用来计算求和的 sigma 函数。当然，main 函数是必不可少的。在 main 函数中调用 sigma 函数进行求和计算，而 sigma 函数又调用 power 函数进行乘方运算，这就是函数的嵌套调用。

参考程序如下：

```
#include <stdio.h>
double sigma(int m, int t);                 // 以下两行为函数原型的声明
double power(int p, int q);
int main( )                                 // 主函数
{
    int k,n;
    double sum;
    scanf("%d%d",&k,&n);                    // 调用系统函数输入两个数
    sum=sigma(n,k);                         // 调用 sigma 函数，把求和结果存到 sum 中
    printf("%d\n",sum);                     // 调用系统函数输出结果
    return 0;
}

double sigma(int m, int t)                  // 求和函数的定义
{
    int i;
    double p,sum=0;
    for (i=1; i<=m; i++)
    {
        p=power(i,t);                       // 调用 power 函数，得到乘方的结果
        sum=sum+p;
    }
    return sum;                             // 把结果传回主调函数 main 中
}
double power(int p,int q)                   // 乘方函数的定义
{
```

```
    int i;
    double product=1;
    for (i=1;i<=q;i++)
        product=product*p;
    return product;                    // 把结果传回主调函数 sigma 中
}
```

可以看出，sigma 函数和 power 函数是两个相互独立、互不隶属的函数。由于这两个函数会被调用，需对其作外部声明。程序从 main 函数开始执行，在执行过程中，调用了 sigma 函数进行求和运算，而 sigma 函数在被调用过程中又调用了 power 函数进行乘方运算，这就是函数的嵌套调用。

3.5.2　函数的递归调用

函数在被调用的过程中，又直接或间接地调用自身，则称函数的递归调用。这种函数也称为递归函数。例如有如下函数：

```
double func (double x)
{
    double y;
    y=2*func(x-1);
    return y;
}
```

这是一个递归函数，但是运行该函数将无休止地调用其自身，这当然是不正确的。为了防止递归调用无终止地进行，必须在函数内有终止递归调用的方法。常用的办法是加条件判断，满足某种条件后就不再做递归调用，然后逐层返回。

下面举例说明。

【例 3.5】 用递归方法求整数 n 的阶乘 $n!$。

分析：一般，$n!$描述成为：

$$n! = 1\times2\times3\times\cdots\times(n-1)\times n$$

但是，只要稍做变换，就可以将其描述成为：

$$n! = n\times(n-1)\times\cdots\times3\times2\times1 = n\times(n-1)!$$

这样，一个整数的阶乘就被描述成为一个规模较小的阶乘与一个数的积。同理，可以将$(n-1)!$描述成$(n-1)\times(n-2)!$，依次类推。于是，一个问题就被描述成一个较小规模的同样类型的问题了。这种关系可用如下递归公式表示：

$$n! = \begin{cases} 1 & (n = 0,1) \\ n\times(n-1)! & (n\geqslant 2) \end{cases}$$

用递归方法实现的参考程序如下：

```
#include <stdio.h>
double fact(int n);                    // 函数原型声明
int main( )
{
    int    n;
    double result;
    scanf("%d",&n);
    result=fact(n);
    if (result==-1)                    // 返回-1，表明输入不正确
        printf("Input error!\n");
    else
        printf("%d!= %.0f\n",n,result);
    return 0;
}
double fact(int n)                     // 递归函数，计算 n!
```

```
{
    if (n < 0)                          // n<0，返回−1
        return −1;
    else if(n==0 || n==1)               // 递归终止条件
        return 1;
    else
        return n*fact(n−1);             // 递归调用
}
```

其求解过程是一个如图 3.2 所示的调用？回代过程。

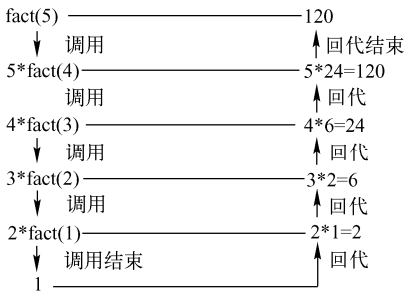

图 3.2　函数的递归调用

在递归调用过程中，每次调用都将问题用较小规模的问题描述代替，直到问题的描述小到可以直接给出解为止，接着便开始一个回代过程。回代过程是从一个已知值推出下一个值的过程。

在图 3.2 中，fact(n)的返回值是 n×fact(n−1)，而求 fact(n−1)的值，需要 fact(n−2)……直到最后找到 fact(1)为止。例如 n=5 时，返回值是 5×fact(4)，而 fack(4)调用的返回值是 4×fact(3)，fact(3) 调用的返回值是 3×fact(2)，fact(2)调用的返回值是 2×fact(1)。fact(1)的返回值是 1，是一个已知数，回过头根据 fact(1)求出 fact(2)，将 fact(2)的值乘以 3 得到 fact(3)，将 fact(3)的值乘以 4 得到 fact(4)，再将 fact(4)的值乘以 5，得到 fact(5)。最终结果由 printf 函数输出。

递归调用不应无限制地进行下去，当调用有限次以后，就应到达递归调用的终点，得到一个确定值（如本例中的 fact(1)=1），然后进行回代。在这样的递归程序中，程序员要写清楚调用结束的条件，以保证程序不会无休止地调用。任何有意义的递归总是具备两个特性：①存在限制条件，当符合这个条件时，递归便不再继续；②每次递归调用之后越来越接近这个限制条件。

需要注意的是，大量的递归调用会占用很多时间和额外的内存。其实，任何能够用递归解决的问题都能用迭代的方法解决。就像本例一样，用迭代的方法更容易理解、效率更高。只有在那些使用递归能够更自然地反映要求解问题的过程时才考虑选用递归方法。

3.6　库　函　数

所谓库函数，就是系统提供的可以实现某种功能的函数的集合。ANSI 标准定义了 C 语言的标准库函数，如数学类函数、输入/输出类函数、字符处理类函数、图形类函数和时间日期类函数等，其中每一类又包括几十到上百种函数。比如在前面经常使用的 scanf 函数、printf 函数，就是输入/输出类函数。一般的 C 语言编译环境都提供了对这些库函数的支持，需要时可以查阅相关文档。

库函数的方便之处在于，系统已定义好，用户可以直接使用它们。比如若想用 printf 函数打印输出，只要了解该函数的功能、参数的个数和类型及函数返回值，具体使用时按照给定参数调用 printf 函数即可。

每一类库函数都有一个相应的头文件，该头文件包含了该库中所有函数的函数原型，以及这些函数所需的各种数据类型和常量的定义，这些头文件的扩展名一般为“.h”。在调用这些库函数时，需要在当前源文件的开始处添加#include 命令。例如，如果要使用数学函数 sqrt 进行开方运算，就需要在源文件中增加一行：

```
#include <math.h>  或者  #include "math.h"
```

而且这行预处理指令必须放到源文件的头部。#include 命令包含头文件的这两种方式的区别详见第 7 章。

其实，C 语言的库函数并不是 C 语言本身的一部分，它是由编译程序根据一般用户的需要编制并提供用户使用的一组程序。C 语言的库函数极大地方便了用户，同时也补充了 C 语言本身的不足。

在 C 语言程序的开发过程中，应当尽可能地使用库函数提供的功能，这样既可以提高程序的运行效率，又可以提高编程的质量。在使用库函数时，应清楚以下 4 个方面的内容：

① 函数的功能及所能完成的操作；

② 参数的个数和类型，以及每个参数的意义；

③ 返回值的意义及类型；

④ 需要使用的头文件。

3.7　变量的作用域与存储类型

C 语言的变量要先定义后使用。在定义变量时，需要指定变量的数据类型及变量名，编译器会给变量分配相应的内存空间。前面我们更多关注的是变量的数据类型，但变量的作用域和变量的存储类型等问题并未描述，本节将对此进行介绍。

3.7.1　变量的作用域

在 C 语言中，所有的变量都有自己的作用域。所谓变量的作用域是指该变量有效的区域。变量定义的位置不同，其作用域也不同。作用域是从空间角度对变量特性的一个描述。按照变量的作用域，将 C 语言中的变量分为局部变量和全局变量。

1．局部变量

定义变量可以在函数的开头、函数内的复合语句内及函数的外部。在一个函数内部定义的变量只在本函数范围内有效，在此函数以外是不能使用这些变量的。在复合语句内定义的变量只在本复合语句范围内有效，只有在本复合语句内才能引用它们。把在函数内部和复合语句内部定义的变量称为局部变量（也称内部变量）。

C 语言规定，形参只在函数内有效。因此形参虽然定义在函数名后面的括号内，在函数体花括号之外，但也属于函数内部的局部变量，其作用域被限定在定义它的函数内部。

例如：

```
int func (int a)
{
    int b,c;
    …
}

int main( )
{
    int d;
    …
    {
        int d, i, j;
        …
    }
    {
        int i, j
        …
    }
    return 0;
}
```

a,b,c 的作用域

1

2

d,i,j 的作用域

3

i,j 的作用域

d 的作用域

在 func 函数中定义了变量 b 和 c，它们的作用域限于 func 函数内。同时 func 函数还有个形参 a，其作用域也限于 func 函数内。同理，变量 d、i、j 的作用域限于 main 函数内。

由于变量作用域范围的限定，同一个代码块（位于一对花括号之间的语句）内不可以定义同名变量，不同代码块内可以定义同名变量。例如，标号 2 处定义的 i、j 和标号 3 处定义的 i、j，由于处于不同的代码块内且彼此并不嵌套，所以它们互不干扰。但是当代码块处于嵌套状态时，如果内层代码块有一个变量的名字与外层代码块的变量同名，则内层的变量会屏蔽外层的变量，也就是说，此时在标号 2 内所使用的变量 d 系统认为是该复合语句内定义的变量 d，而非 main 函数开始处第 1 行定义的变量 d。当然，在实际应用中，应避免在嵌套的代码块中出现相同的变量名。

说明：

① 主函数中定义的变量也是局部变量，只能在主函数中使用，不能在其他函数中使用。同时，主函数中也不能使用其他函数中定义的变量。因为主函数也是一个函数，与其他函数是平行关系。

② 形参变量是属于被调函数的局部变量，实参变量是属于主调函数的局部变量。

③ 允许在不同函数中使用相同的变量名，它们代表不同的对象，分配不同的内存单元，互不干扰，也不会发生混淆。

④ 在复合语句中也可定义变量，其作用域只在定义它的复合语句范围内。

⑤ 变量的定义必须在可执行语句之前。进入{}，首先是定义语句，然后才是执行语句。

【例 3.6】 如下程序代码是实现两个整数相加的功能，分析程序中用到的变量作用域范围。

```
#include <stdio.h>
int plus(int x, int y)
{
    return x+y;
}
int main( )
{
    int x,y,result;
    scanf("%d%d",&x,&y);
    result=plus(x,y);
    printf("%d+%d=%d\n",x,y,result);
    return 0;
}
```

plus 函数和 main 函数中都有同名的变量 x 和 y。由于它们属于各自所在的代码块，所以彼此互不影响。plus 函数中的 x 和 y 是形参，它们只在 plus 函数被调用时起作用，调用结束时就失效；main 函数中定义的 x 和 y 是实参，它们也只在 main 函数中起作用。

2．全局变量

定义在函数外部的变量称为全局变量（也称外部变量）。全局变量定义在所有函数之外，不属于任何函数或代码块，它属于整个源文件。全局变量的有效范围从定义变量的位置开始到本源文件结束。

```
int a,b;                    // a、b 的作用范围从该位置开始到整个源文件结束
int func1(int c)
{
    int d;                  // 局部变量
    …
}
int e, f;                   // e、f 的作用范围从该位置开始到整个源文件结束
int func2( )
{
```

```
    int a,b;                    // 局部变量
    …
}
int main( )
{
    int g,h;                    // 局部变量
    …
}
```

a、b、e、f 都是在函数外部定义的全局变量，但它们的作用范围不同。在 main 函数和 func2 函数中可以使用全局变量 a、b、e、f，但函数 func1 中只能使用 a、b，而不能使用 e、f。需要注意的是，全局变量可以与局部变量同名，但在局部变量起作用的范围内，全局变量被屏蔽，即全局变量不起作用。例如，全局变量 a、b 在 func2 函数中被屏蔽，因为 func2 函数中有自己的局部变量 a、b。

说明：

① 全局变量可加强函数模块之间的数据联系。由于在同一文件中所有函数都能使用全局变量，所以可以利用全局变量从函数中得到一个以上的值。但是全局变量又会使函数依赖这些值，从而使得函数的独立性及可移植性降低。从模块化程序设计的观点来看，这是不利的，因此尽量不要使用全局变量。

② 全局变量在程序的执行过程中一直占用内存，可以随时访问。

③ 全局变量在定义时如果没有初始化，它们会被系统自动初始化为零，而局部变量在定义时不会自动初始化。

【例 3.7】 利用全局变量编写一个函数，求某班级学生成绩的最高分、最低分和平均分。

分析：本题要求编写一个函数求学生成绩的最高分、最低分和平均分三个值。根据前面的讲述可知，一个函数最多只有一个返回值。解决这类问题的方法有多种，在这里通过全局变量来实现。针对本例编写函数 average 返回平均值，最高分和最低分通过定义两个全局变量得到。

```
#include   <stdio.h>
double max=0,min=100;                   // 定义全局变量
double average(int n);                  // 声明函数 average
int main( )
{
    int m;
    double ave ;
    scanf("%d",&m);                     // 输入班级学生人数
    ave=average(m);                     // 调用 average 函数
    printf("ave=%6.2f,max=%6.2f,min=%6.2f\n",ave,max,min);
    return 0;
}
double average(int n)                   // 形参 n 用于接收班级的人数
{
    int i;
    double s,ave,sum=0;
    for (i=1;i<=n;i++)
    {
        scanf("%lf",&s);
        if (s>max)
            max=s;
```

```
            if (s<min)
                min=s;
            sum=sum+s;
        }
        ave=sum/n;
        return ave;
    }
```

main 函数通过调用 average 函数把实参 m（学生人数）传递给形参 n。average 函数把计算所得平均分 ave 通过 return 语句带回。这样，在 main 函数中就得到了平均分，而最高分和最低分是通过全局变量 max 和 min 获得的。max 和 min 是全局变量，各个函数都可以直接引用或修改它们的值。由于 average 函数把最高分和最低分分别存放在 max 和 min 中，这样 max 和 min 的初始值就被修改了，而修改后的值可以被 main 函数使用。因此，在 main 函数中也可以直接输出 max 和 min 的值，以得到最高分和最低分。

3.7.2　变量的存储类型

所谓变量的存储类型是指存储变量值的内存类型。变量的存储类型决定变量何时创建、何时销毁及它的值将保持多久。变量的存储类型可分为静态存储和动态存储两种。

静态存储变量通常是在变量定义时就分配存储单元并一直保持不变，直到整个程序结束。3.7.1 节中介绍的全局变量即属于此类存储方式。

动态存储变量是在程序执行过程中，使用到它时才分配存储单元，使用完毕立即释放。典型的例子是函数的形式参数。在函数定义时并不给形参分配存储单元，只有在函数被调用时，才予以分配，调用函数完毕立即释放。如果一个函数被多次调用，则反复地分配、释放形参变量的存储单元。

由以上分析可知，静态存储变量是一直存在的，而动态存储变量则时而存在、时而消失。把这种由于变量存储方式不同而产生的特性称为变量的生存期。生存期表示了变量存在的时间。生存期和作用域分别从时间和空间这两个不同的角度来描述变量的特性，两者既有联系，又有区别。一个变量究竟属于哪一种存储方式，并不能仅从其作用域来判断，还应该有明确的存储类型说明。

在 C 语言中，对变量的存储类型说明包括 4 种：自动变量（auto）、寄存器变量（register）、静态变量（static）和外部变量（extern）。

在介绍了变量的存储类型之后，对一个变量的定义就不仅应说明其数据类型，还应说明其存储类型。因此变量定义的完整形式应为：

```
存储类型　数据类型　变量名;
```

下面分别进行介绍。

1．自动变量（auto 变量）

这种存储类型是 C 语言程序中使用最广泛的一种类型。C 语言规定，函数内凡未加存储类型说明的变量均视为自动变量，也就是说自动变量可省去说明符 auto。在前面各章的程序中所定义的变量凡未加存储类型说明符的都是自动变量。例如：

```
int x,y;　等价于　auto int x,y;
```

说明：

① 自动变量的作用域仅限于定义该变量的代码块内。在函数中定义的自动变量，只在该函数内有效。在复合语句中定义的自动变量只在该复合语句中有效。

② 当程序执行到定义自动变量的代码块时，该自动变量才被创建，当程序的执行离开该代码块时，这些自动变量便自行销毁，所以自动变量属于动态存储方式。

③ 函数的形式参数默认也是自动变量。

2．寄存器变量（register 变量）

一般情况下，变量都存放在内存中，当对一个变量频繁读写时，必须要反复访问内存，从而花费大量的存取时间。为此，C 语言提供了另一种变量，即寄存器变量。这种变量存放在 CPU 的寄存器中，使用时不需要访问内存，而直接从寄存器中读写，这样可以提高效率。寄存器变量的说明符是 register。对于循环次数较多的循环控制变量以及循环体内反复使用的变量均可定义为寄存器变量。例如：

```
register int x,y;
```

说明：

① 只有局部自动变量和形式参数才可以定义为寄存器变量。因为寄存器变量属于动态存储方式，所以凡是需要采用静态存储方式的变量不能定义为寄存器变量。

② 由于 CPU 中寄存器的个数是有限的，所以编译器可以忽略 register 关键字。当有太多的变量声明为 register 时，只有个别变量会存储在寄存器中，其余的编译器会按自动变量处理。其实，编译器都有自己的寄存器优化方案，它会决定哪些变量存储于寄存器中，所以在变量的定义中，一般并不需要使用 register 关键字。

3．静态变量（static 变量）

相对于变量的动态存储而言，关键字 static 用来声明静态存储的变量。

对于局部变量，static 关键字用于修改变量的存储类型，但该变量的作用域不受影响，仍然局限于定义它的函数或复合语句内。静态局部变量有如下两个使用特性：

① 静态局部变量在程序执行之前创建，并在程序整个执行期间一直存在。

② 静态局部变量仅赋一次初值。在编译时系统即为静态局部变量分配内存空间，并赋初值。在随后的程序运行过程中，当自定义函数调用结束后，静态局部变量仍将保持函数调用结束时的值。当该函数再次被调用时，静态局部变量不再重新被赋初值，而是保留上次调用后的值。

【例 3.8】 编写一个函数实现值自增 1。在主函数中循环调用 3 次该函数，并输出结果。考查静态局部变量的值。

参考程序如下：

```
// f1.c
#include <stdio.h>
int add( );
int main( )
{
    int i,result;
    for (i=1;i<=3;i++)
    {
        result=add( );
        printf("%–4d",result);
    }
    return 0;
}
int add( )
{
    auto int num=5;   // 自动局部变量
    num++;
    return num;
}
```

```
// f2.c
#include <stdio.h>
int add( );
int main( )
{
    int i,result;
    for (i=1;i<=3;i++)
    {
        result=add( );
        printf("%–4d",result);
    }
    return 0;
}
int add( )
{
    static int num=5;   // 静态局部变量
    num++;
    return num;
}
```

以上的两个源文件中，只有函数 add 里的变量声明有所不同，一个是自动存储类型，一个是静态

存储类型。对于 f1.c 文件，输出结果为“6　6　6”。这是因为每次 add 函数被调用时，系统都要为自动局部变量 num 分配存储空间，调用结束后释放空间，所以 num 的初始值每次都是 5，然后加 1 得 6。

对于 f2.c 文件，输出结果为“6　7　8”。这是由于变量 num 是静态局部变量，在程序运行前就被初始化为 5，且只做这一次初始化，以后每次 add 函数被调用时，num 都是使用上次调用后的结果值。所以，当第 1 次调用 add 时，num 初始值为 5，然后加 1，输出为 6；当第 2 次调用时，就不初始化了，这时 num 的值为上次的 6，然后加 1，输出 7；当第 3 次调用时，num 为 7，加 1 就是 8 了。

说明：

① 静态局部变量属于静态存储方式。

② 静态局部变量在函数内定义，即使函数被多次调用也只进行一次赋初值操作，能保留最后的运行结果。静态局部变量始终占用内存空间，直到整个程序运行结束才释放。

③ 静态局部变量虽然在整个源程序中都存在，但是其作用域仍与自动变量相同，即只能在定义该变量的函数内使用该变量。退出该函数后，尽管该变量还继续存在，但不能使用它。

④ 静态局部变量若在声明时未赋初值，则系统自动初始化为 0（对数值型变量）或'\0'（对字符变量）。而对自动变量来说，不赋初值则其值是不确定的。

根据静态局部变量的特点可以看出，它的生存期为整个程序运行期间。虽然离开定义它的函数后该变量不能使用，但若再次调用定义它的函数时，它又可以继续使用，而且保存了前次被调用后留下的值。因此，当多次调用一个函数且要求在调用之间保留某些变量的值时，可以采用静态局部变量。虽然用全局变量也可以达到上述目的，但全局变量有时会造成意外的副作用，因此仍以采用静态局部变量为宜。

static 除了可以用于局部变量的声明外，还可以用于全局变量的声明。在全局变量的声明之前加上 static 关键字就构成了静态全局变量。全局变量本身就是静态存储方式，静态全局变量当然也是静态存储方式。static 关键字并没有改变全局变量的存储类型，而是对其作用域进行了限制。例如，当一个程序由多个源文件组成时，非静态全局变量在各个源文件中都是有效的。而静态全局变量则只能在定义该变量的源文件内有效，在同一程序的其他源文件中不可见，也不能使用。从以上分析可以看出，把局部变量用 static 声明后改变了该变量的存储方式，但作用域并没有变，而把全局变量用 static 声明后改变了它的作用域，限制了它的使用范围，但存储方式并没有变。因此，static 关键字用于不同的地方，具有不同的含义，使用时一定要注意。

4．外部变量（extern 变量）

外部变量是在函数的外部定义的全局变量。外部变量和全局变量是对同一类变量的两种不同角度的提法。全局变量是从它的作用域提出的，外部变量是从它的存储方式提出的。

3.7.1 节中介绍了全局变量从它的定义位置开始一直到整个源文件结束的范围内，所有函数都可以使用该变量。但如果要在全局变量的定义点之前使用该全局变量，就应该在引用之前用关键字 extern 对该变量做外部变量声明。这样就可以扩展全局变量的使用范围。

一般来说，外部变量有以下两种用法：

① 在同一个文件内，为了使全局变量在定义点之前的函数中也能使用，在函数中用 extern 加以声明。

【例 3.9】 编写一个函数求圆的面积。

参考程序如下：

```
#include <stdio.h>
double area(double r)                  // 计算圆的面积
{
    extern double pi;                  // 外部变量声明，把全局变量 p 的作用域扩展到从此处开始
```

```
    return pi*r*r;
}
double pi=3.141593;                    // 定义全局变量 p 并初始化，其作用域从此往下有效
int main( )
{
    double r;
    scanf("%lf",&r);                   // 输入圆的半径
    printf("area=%f\n",area(r));
    return 0;
}
```

这个例子很简单，主要用来说明使用外部变量的方法。由于全局变量 p 的定义位置出现在 area 函数的后面，本来 area 函数是不能引用变量 p 的，但通过在 area 函数中用 extern 对变量 p 做外部变量声明，即可把 p 的作用域扩展到该位置。这样，在 area 函数中也可以合法地使用全局变量 p 了。

建议把外部变量的定义放在引用它的所有函数之前，这样可以避免再进行声明。

② 当一个源程序由若干个源文件组成时，在一个源文件中定义的外部变量可以在其他源文件中使用 extern 进行声明，使其在其他源文件中也可以被访问。

例如，有一个源程序由源文件 f1.c 和 f2.c 组成：

```
// f1.c 文件
int a,b;                               // 外部变量定义
char c;                                // 外部变量定义
int main( )
{
    …
}
// f2.c 文件
extern int a,b;                        // 外部变量声明
extern char c;                         // 外部变量声明
int func(int x, int y)
{
    …
}
```

在 f1.c 和 f2.c 两个文件中都要使用 a、b、c 3 个变量。在 f1.c 文件中把 a、b、c 都定义为外部变量。在 f2.c 文件中用 extern 把 3 个变量声明为外部变量，表示这些变量已在其他文件中定义，编译系统不再为它们分配内存空间。这样就可以把一个文件中定义的外部变量的作用域扩展到另一个文件中。

3.8　指针与函数

指针是 C 语言中一个重要的概念，也是 C 语言的特色，通过在程序中运用指针，可以使程序变得简洁而高效，同时也可以实现许多复杂的功能。在第 2 章已经介绍了指针的概念及指针的基本使用方法。下面进一步介绍指针与函数的相关知识。

3.8.1　指针作为函数参数

函数的参数可以是前面学过的简单数据类型，也可以是指针类型，那么用指针变量作为函数参数与普通变量作为函数参数有什么不同呢？下面通过一个例子来说明。

【例 3.10】 输入 a 和 b 两个整数，用自定义函数实现这两个值的交换，并在主函数中输出交换后的结果。

分析：完成两个数的交换算法本身很简单，现在要通过自定义函数的方式实现，即在主调函数中传递两个需交换的值，通过被调函数的执行把这两个数交换后的结果反映到主调函数中。现在以普通

变量作为函数的形参，看运行结果是否能满足要求。

参考程序如下：

```
#include <stdio.h>
void swap(int x, int y)                          // 定义 swap 函数，实现两个数的交换
{
    int temp;
    temp=x;
    x=y;
    y=temp;
}
int main( )
{
    int a,b;                                     // 定义两个需要交换的整数
    scanf("%d%d",&a,&b);                         // 输入两个整数
    printf("交换前: a=%d,b=%d\n",a,b);           // 调用 swap 函数前，先输出这两个数
    swap(a,b);                                   // 调用 swap 函数，期望实现两个数交换
    printf("交换后: a=%d,b=%d\n",a,b);           // 调用 swap 函数后，输出结果
    return 0;
}
```

运行结果如下：

```
请输入两个整数：3  5↙
交换前: a=3,b=5
交换后: a=3,b=5
```

很明显，程序运行的结果并没有达到要求，a 和 b 的值没有交换，问题出在哪里呢？下面从 swap 函数入手分析：该函数定义了两个 int 类型的形参，没有返回值，函数体完成了两个形参的交换，但最后主调函数并没有得到这种交换后的结果。为了验证 swap 函数确实完成了两个形参之间的交换，可以在 swap 函数最后再增加一条输出语句：

```
printf("x=%d,y=%d\n",x,y);
```

重新运行这个程序，会发现输出结果为“x = 5, y = 3”。x 和 y 的值确实交换了，但交换后的结果没有返回到主函数的 a 和 b 中。首先可以明确，一个函数通过 return 语句最多只能返回一个值，而 swap 函数的作用是把交换后的两个形参的值再返回到主调函数中，所以在 swap 函数中不可能用 return 语句实现，而只能考虑通过形参完成该功能。但为什么定义的两个形参没有把交换后的结果返回到主函数中呢？通过函数知识的学习可知，实参到形参的数据传递是单向的“值传递”，只能是实参把值传给形参，而不能由形参传给实参。并且形参和实参占用不同的内存单元，只有当函数被调用时，形参才被临时分配内存空间，然后接收实参传递过来的值进行有关运算。当调用结束后，形参占用的内存空间被释放，形参的值也将丢失。因此，形参值的改变并不会影响到主调函数的实参值。该过程如图 3.3 所示。

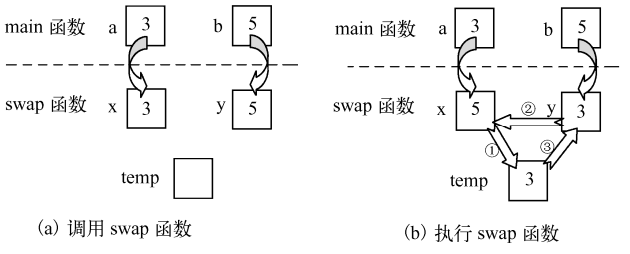

图 3.3　函数调用执行过程示意图

那么，如何才能通过被调函数的执行实现主调函数中值的修改呢？答案是用指针变量作为函数参数。

【例 3.11】 输入 a 和 b 两个整数，用自定义函数实现这两个值的交换，并在主函数中输出交换后的结果。用指针类型的数据作函数参数。

参考程序如下：

```
#include <stdio.h>
```

```
void swap(int *x, int *y)                    // 定义 swap 函数，用指针变量做形参
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}
int main( )
{
    int a,b;                                 // 定义两个需要交换的整数
    int *p1,*p2;                             // 定义两个指向 int 类型的指针变量
    scanf("%d%d",&a,&b);                     // 输入两个整数
    printf("交换前: a=%d,b=%d\n",a,b);       // 调用 swap 函数前，先输出这两个数
    p1=&a;                                   // 使 p1 指向 a
    p2=&b;                                   // 使 p2 指向 b
    swap(p1,p2);                             // 调用 swap 函数，实现两个数交换
    printf("交换后: a=%d,b=%d\n",a,b);       // 调用 swap 函数后，输出结果
    return 0;
}
```

运行结果如下：

```
请输入两个整数: 3 5↙
交换前: a=3,b=5
交换后: a=5,b=3
```

从本例的运行结果可以看出，用指针变量作为函数参数后，确实实现了主调函数中两个值的互换，这又是为什么呢？下面分析该程序。

在主函数中，变量 a 和 b 的地址分别赋给了指针变量 p1 和 p2，并用指针变量 p1 和 p2 作为函数实参去调用 swap 函数。在 swap 函数中用指向整型的指针变量 x 和 y 作为函数形参。当函数调用时，p1 的值（即变量 a 的地址）传给了指针变量 x，于是 x 指向了 a，*x 是取 x 所指向的变量的值，即 a 的值。同理，p2 的值（即变量 b 的地址）传给了指针变量 y，于是 y 指向了 b，*y 就是 b 的值。因此，在执行 swap 函数时，借助中间变量 temp 对*x 和*y 进行值的交换操作，实际上是对 x 和 y 所指向的变量 a 和 b 的值进行的交换。所以，用指针变量作为函数参数可以实现题目要求。其过程如图 3.4 所示。

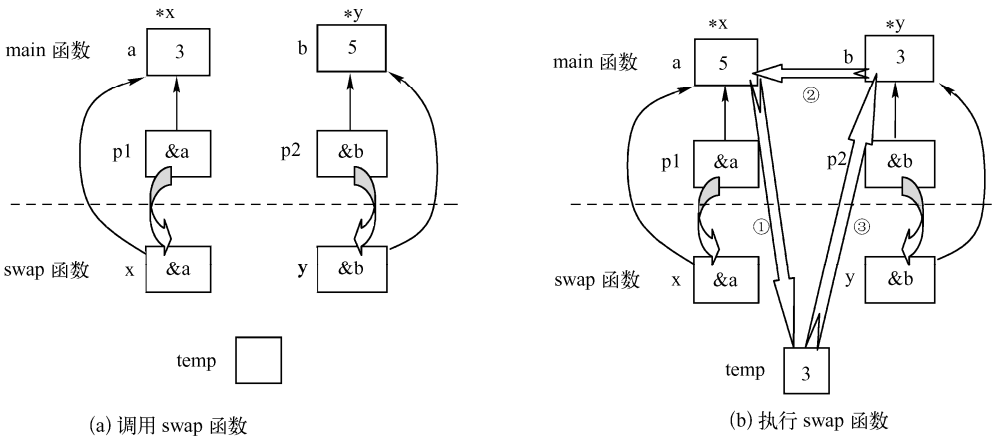

图 3.4　指针变量作为函数参数执行过程示意图

其实，指针变量作实参时，与普通变量一样，也是采用单向的值传递方式，即将指针变量的值（该值是一个地址）传递给被调函数的形参（必须也是一个指针变量）。由于形参接收的实参传递的值

是一个地址，这样两者就指向了同样的变量，因此在被调函数中对形参所指向的变量的修改，是以一种间接访问的方式来访问主调函数中的普通变量。这一点编写程序时必须清楚。

这个例子告诉我们，如果想用 C 语言的函数调用实现多个值的改变，可以设计一个形参是指针变量的被调函数，在主调函数中将变量的地址作为实参传递给形参指针，在被调函数中就可以通过间接访问的方式改变该地址中的值，从而使被调函数实际返回多个值到主调函数中。

当函数形参是指针变量时，实参除了可以是同类型的指针变量外，也可以是地址常量，即普通变量取其地址。例如：

```
void swap(int *x, int *y);
int main( )
{
    int a,b;
    scanf("%d%d",&a,&b);
    printf("交换前: a=%d,b=%d\n",a,b);
    swap(&a,&b);                    // 调用 swap 函数，变量 a 和 b 的地址常量作为实参
    printf("交换后: a=%d,b=%d\n",a,b);
    return 0;
}
```

需要强调的是，用指针变量作为函数参数时，在函数内部，形参指针的使用方式通常都应该是间接访问的方式，即：

```
temp=*x;   *x=*y;   *y=temp;
```

通过这种形式来改变形参指针所指向的主调函数中普通变量的值。

切记在函数内部不能直接修改指针变量的值，一旦修改了，表示指针变量的指向发生了改变，不再指向原来的变量，这对所指向的变量的存储单元内的值无任何影响。例如，如果将例 3.11 中的 swap 函数改成如下形式：

```
void swap(int *x, int *y)
{
    int *temp;                      // 定义指针变量 temp
    temp=x;                         // 以下 3 句完成形参指针的交换
    x=y;
    y=temp;
}
```

重新运行程序，可以发现结果和期望的不一样，原因是在 swap 函数中只交换了指针变量 x 和 y 的指向，并没有交换这两个指针所指向的变量的值。因此，主调函数中变量 a 和 b 并没有被改变。

3.8.2　返回指针值的函数

在 C 语言中一个函数的返回值可以是整型值、字符值、实型值等，也可以是指针（即地址）。这种返回指针值的函数的一般定义形式为：

```
数据类型 *函数名(形参列表)
{
    函数体;
}
```

【例 3.12】 编写一个函数求某班级学生成绩的平均分，用指针实现。

分析：该程序在讲全局变量时已经出现过，这个例子可以不用下面这种方式实现，这里仅是用其说明返回指针值的函数的使用。

```
#include <stdio.h>
double *average(int n);                 // 函数原型声明
int main( )
{
```

```
    int m;
    double *ave;
    scanf("%d",&m);                    // 输入班级学生人数
    ave=average(m);                    // 调用 average 函数
    printf("ave=%6.2f\n",*ave);
    return 0;
}
double *average(int n)                 // 定义返回指针值的函数
{
    int i;
    double *result;
    double s,sum=0;
    static double ave;

    result=&ave;
    for (i=1;i<=n;i++)
    {
        scanf("%lf",&s);
        sum=sum+s;
    }
    ave=sum/n;
    return result;                     // 返回指针值
}
```

函数 average 是返回指针值的函数。函数的返回值仍然通过 return 返回，只是返回的数据类型是指针类型。在本例中，通过函数的返回值得到保存平均分的变量的地址，进而得到平均分。

3.8.3　指向函数的指针变量

在 C 语言中，一个函数总是占用一段连续的内存区，而函数名表示该函数所占内存区的首地址（或称入口地址）。可以把函数的这个首地址赋予一个指针变量，使该指针变量指向该函数，然后通过指针变量即可找到并调用这个函数。把这种指向函数的指针变量称为函数指针变量。

函数指针变量定义的一般形式为：

```
数据类型　(*指针变量名)(函数形参列表);
```

“数据类型”说明函数的返回类型，由于( )的优先级高于*，所以指针变量名外的括号必不可少，后面的“函数形参列表”表示指针变量指向的函数所带的参数列表。例如：

```
int func(int x);    // 声明一个函数
int (*f)(int x);    // 声明一个指向函数的指针变量
f=func;             // 将 func 函数的首地址赋给指针 f
```

赋值时仅使用函数名称，不带括号，也不带参数。由于 func 代表函数的首地址，因此经过赋值以后，指针 f 就指向函数 func 的首地址。

调用时，指针变量需用间接访问的形式，同时应加括号，例如：

```
(*f)(实参列表)
```

【例 3.13】 编写一个函数求整数 a 和 b 中的较大者。

```
#include <stdio.h>
int max(int x,int y)
{
    return(x>y?x:y);
}
int main( )
{
    int a,b,c;
    int (*p)(int,int);                 // 定义指向函数的指针变量
```

```
    scanf("%d%d",&a,&b);
    p=max;                              // 使 p 指向 max 函数
    c=(*p)(a,b);                        // 通过指针变量调用 max 函数，相当于调用 max(a,b)
    printf("a=%d,b=%d,max=%d\n",a,b,c);
    return 0;
}
```

p 是指向函数的指针变量，所以可把函数 max 赋给 p，即把 max 函数的入口地址赋给 p，以后就可以用 p 来间接调用该函数。实际上 p 和 max 都指向同一个入口地址，不同的是 p 是一个指针变量，它可以指向与它声明相符合的任何函数，不像函数名那样是固定不变的。

这样，如果想调用一个函数，除了可以通过函数名调用以外，还可以通过指向函数的指针变量来调用该函数。

3.9　典 型 例 题

【例 3.14】 用函数实现牛顿迭代法求一元三次方程的根。

分析：牛顿迭代法的公式是：$x = x_0 - f(x) / f'(x)$，设迭代误差小于 10^{-5} 时结束。

参考程序如下：

```
#include <stdio.h>
#include <math.h>
double solut(double a,double b,double c,double d)
{
    double x=1,x0,f,f1;
    do
    {
        x0=x;
        f=((a*x0+b)*x0+c)*x0+d;
        f1=(3*a*x0+2*b)*x0+c;
        x=x0-f/f1;
    }while(fabs(x-x0)>=1e-5);
    return x;
}
int main( )
{
    double a,b,c,d;
    scanf("%lf%lf%lf%lf",&a,&b,&c,&d);          // 输入一元三次方程的系数
    printf("一元三次方程为：%5.2fx^3+%5.2fx^2+%5.2fx+%5.2f=0\n",a,b,c,d);
    printf("该方程的根为：x=%7.2f\n",solut(a,b,c,d));
    return 0;
}
```

【例 3.15】 输入某年、某月、某日，编写函数判断这一天是这一年的第几天。

分析：以 3 月 5 日为例，应该先把前两个月的天数加起来，然后再加上 5 即为本年的第几天，特殊情况，闰年且输入月份大于 3 时需考虑多加 1 天。

参考程序如下：

```
#include <stdio.h>
int mon(int year,int month,int day);
int leap_year(int year);
int main( )
{
    int day,month,year,sum;
    scanf("%d%d%d",&year,&month,&day);      // 输入年、月、日
    sum=mon(year,month,day);
```

```
    printf("It is the %dth day.\n", sum);
    return 0;
}
int mon(int year, int month, int day)
{
    int sum,leap;
    switch(month)                                   // 先计算某月以前月份的总天数
    {
        case 1:   sum=0;   break;
        case 2:   sum=31;  break;
        case 3:   sum=59;  break;
        case 4:   sum=90;  break;
        case 5:   sum=120; break;
        case 6:   sum=151; break;
        case 7:   sum=181; break;
        case 8:   sum=212; break;
        case 9:   sum=243; break;
        case 10:  sum=273; break;
        case 11:  sum=304; break;
        case 12:  sum=334; break;
        default:  printf("data error\n"); break;
    }
    sum=sum+day;                                    // 再加上某天的天数
    leap=leap_year(year);
    if (leap==1&&month>2)                           // 如果是闰年且月份大于 2，总天数应该加 1 天
        sum++;
    return sum;
}
int leap_year(int year)
{
    if (year%400==0 || (year%4==0 && year%100!=0))          // 判断是不是闰年
        return 1;
    else
        return 0;
}
```

【例 3.16】 输入 3 个整数，编写函数利用指针方法将这 3 个数按从小到大排序。

分析：注意交换顺序。

参考程序如下：

```
#include <stdio.h>
void swap(int *p1, int *p2);
int main( )
{
    int n1,n2,n3;
    int *p1,*p2,*p3;
    scanf("%d%d%d",&n1,&n2,&n3);
    p1=&n1;
    p2=&n2;
    p3=&n3;
    if (n1>n2)
        swap(p1,p2);
    if (n1>n3)
        swap(p1,p3);
    if (n2>n3)
        swap(p2,p3);
    printf("The sorted numbers are: %d, %d, %d\n",n1,n2,n3);
    return 0;
```

```
}
void swap(int *x,int *y)
{
    int temp;
    temp=*x;
    *x=*y;
    *y=temp;
}
```

【例 3.17】 编写函数，输入 *n* 为偶数时，调用函数计算 1/2+1/4+…+1/*n*；输入 *n* 为奇数时，调用函数计算 1/1+1/3+…+1/*n*。

参考程序如下：

```
#include <stdio.h>
double peven(int n);
double podd(int n);
int main( )
{
    double sum;
    int n;
    while(1)                          // 保证输入的 n 大于 1
    {
        scanf("%d",&n);
        if (n>1)
            break;
    }
    if (n%2==0)
        sum=peven(n);
    else
        sum=podd(n);
    printf("result=%f\n",sum);
    return 0;
}
double peven(int n)                   // 计算 1/2+1/4+…+1/n
{
    double s=0;
    int i;
    for (i=2;i<=n;i+=2)
        s+=1.0/i;
    return s;
}
double podd(int n)                    // 计算 1/1+1/3+…+1/n
{
    double s=0;
    int i;
    for (i=1;i<=n;i+=2)
        s+=1.0/i;
    return s;
}
```

【例 3.18】 汉诺（Hanoi）塔问题是源于印度的一个古老传说。在世界中心贝拿勒斯（在印度北部）的圣庙里，一块黄铜板上插着三根金刚石柱子。印度教的主神梵天在创造世界的时候，在其中一根针上从下到上地穿好了由大到小的 64 片黄金盘片，这就是所谓的汉诺塔。梵天命令婆罗门把金片按大小顺序重新摆放在另一根柱子上，并且规定，小金片必须放到大金片上面，且在三根柱子之间一次只能移动一个金片。现要求用程序模拟该过程，并输出移动步骤。

分析：这是一个用递归方法解题的典型例子。假设三个柱子用 A、B、C 表示，64 个盘片按从小到大用 1，2，…，64 编号，那么按照规则将 64 个盘片从 A 柱移到 C 柱，这个过程可以用下面三个步

骤表示。

① 将 1 到 63 号盘片从 A 柱移到 B 柱；

② 将第 64 号盘片从 A 柱移到 C 柱；

③ 将 1 到 63 号盘片从 B 柱移到 C 柱。

在上面的三个步骤中，第二步移动一个盘片，只需一步即可完成；第一步和第三步都是移动 63 个盘片，只是移动的源和目标不同。于是问题变为移动这 63 个盘片，而 63 个盘片的移动可以仿照上述步骤进行，问题又转变为移动 62 个盘片。依次类推，直到最后问题转变为移动一个盘片，直接移动即可完成。

经过上面的分析可知，对于移动 n 个盘片的汉诺塔问题，用递归解法可归纳为如下三个步骤：

① 将 n−1 个盘片从 A 柱移到 B 柱；

② 将第 n 号盘片从 A 柱移到 C 柱；

③ 将 n−1 个盘片从 B 柱移到 C 柱。

上面的三个步骤可以归纳为两类操作，移动一个盘片和移动 n−1 个盘片。编写两个函数分别实现这两类操作，用 move 函数实现移动一个盘片，用 Hanoi 函数实现移动 n−1 个盘片。移动一个盘片只需源柱和目标柱，移动 n−1（n−1>1）个盘片除了源柱和目标柱还需要另外一个柱子作为中间过渡，因此两个函数的参数个数不同。具体移动步骤在程序中只能通过信息显示来仿真，用 printf 函数打印输出。

参考程序如下：

```
#include <stdio.h>
void move(int n,char from,char to);
void hanoi(int n,char from,char to,char temp);
int main( )
{
    int n;
    char from,to,temp;                    // 定义三个变量代表三个柱子
    scanf("%d",&n);                       // 输入盘子个数
    from='A'; to='C'; temp='B';
    hanoi(n,from,to,temp);
    return 0;
}
void hanoi(int n,char from,char to,char temp)
{
    if (n==1)
        move(n,from,to);
    else
    {
        hanoi(n-1,from,temp,to);
        move (n,from,to);
        hanoi(n-1,temp,to,from);
    }
}
int count=0;                              // 记录移动次数
void move(int n,char from,char to)
{
    count++;
    printf("第%d 步：%d 号%c-->%c\n",count,n,from,to);
}
```

习　　题

一、选择题

1．C 语言程序由函数组成，下列说法正确的是（　　）。

A）主函数可以在其他函数之前，函数内可以嵌套定义函数

B）主函数可以在其他函数之后，函数内不可以嵌套定义函数

C）主函数必须在其他函数之前，函数内不可以嵌套定义函数

D）主函数必须在其他函数之后，函数内可以嵌套定义函数

2．在 C 语言程序中，若对函数类型未加显示说明，则函数的隐含类型为（　　）。

A）int　　B）double　　C）void　　D）char

3．以下对 C 语言函数的有关描述中，正确的是（　　）。

A）调用函数时，只能把实参的值传给形参，形参的值不能传给实参

B）函数既可以嵌套定义，又可以递归调用

C）函数必须要有返回值

D）程序中有调用关系的所有函数必须放在同一个源程序文件中

4．C 语言规定，调用一个函数时，实参变量和形参变量之间的数据传递是（　　）。

A）由用户指定传递方式

B）由实参传给形参，并由形参传回给实参

C）值传递

D）地址传递

5．关于函数的说法错误的是（　　）。

A）可以在一个函数内定义另一个函数

B）一个函数可以没有参数

C）可根据需要自己来编写函数

D）C 语言用函数实现模块的功能

6．在 C 语言中，函数返回值的类型最终取决于（　　）。

A）函数定义时的函数首部所说明的函数类型

B）return 语句中表达式值的类型

C）调用函数时主调函数所传递的实参类型

D）函数定义时形参的类型

7．以下正确的函数声明是（　　）。

A）int fun(int x,int y)　　B）float fun(int a;int b);

C）double fun(float x,float y);　　D）int fun(int x,y);

8．在一个源文件中定义的全局变量的作用域为（　　）。

A）本文件的全部范围　　B）本程序的全部范围

C）本函数的全部范围　　D）从定义变量的位置开始到本文件结束

9．凡在函数中未指定存储类别的局部变量，其隐含的存储类别为（　　）。

A）自动（auto）　　B）静态（static）

C）外部（extern）　　D）寄存器（register）

10．以下程序的执行结果是（　　）。

```
#include <stdio.h>
int f(int a)
{
    int b=0;
    static int c=3;
    b++;
    c++;
    return(a+b+c);
}
int main( )
{
    int a=2,i;
    for (i=0;i<3;i++)
        printf("%d",f(a));
    return 0;
}
```

A）777　　B）7 7 7　　C）789　　D）7 8 9

11．以下程序的输出是（　　）。

```
#include <stdio.h>
void fun(int a,int b,int c)
{
    a=456; b=567; c=678;
}
int main( )
{
    int x=10, y=20,z=30;
    fun(x,y,z);
    printf("%d, %d, %d\n",x,y,z);
    return 0;
}
```

A）30,20,10　　B）10,20,30　　C）456,567,678　　D）678,567,456

12．若有以下函数：

```
void swap(int *px,int *py)
{
    int *t;
    t=px; px=py; py=t;
}
```

函数调用语句为：“swap(&x,&y);”，则下列说法中正确的是（　　）。

A）交换了*px 和*py 的值

B）主调函数中的变量 x 和 y 交换了地址

C）主调函数中 x 和 y 的值并没有被交换

D）该函数有语法错误，不能执行

二、填空题

1．解决一个复杂问题时自顶向下逐层把系统划分成若干模块的过程称为_____。C 语言程序是用_____来表示模块的。

2．函数调用时，所有参数的传递方式都是_____，即单向传递。

3．为了明确表示一个函数没有返回值，则应该将返回值类型定义为_____。

4．如果被调函数定义在主调函数的后面，则在调用函数前，需要把被调函数的名字、函数类型，以及形参类型、个数和顺序通知编译系统，称为_____。

5．按照变量的作用域，将C语言中的变量分为_____和全局变量。

6．定义函数时，为了能使某个变量的值在下次调用该函数时继续使用，则应将该变量的存储类型定义为_____。

7．swap 函数的定义形式为：

```
void swap(int *pa, int *pb)
{
    int temp;
    temp=*pa;
    *pa=*pb;
    *pb=temp;
}
```

为了能使主调函数中 int 型变量 a、b 的值互相交换，请补充下面的函数调用语句。

```
swap(_____);
```

8．请定义一个指针变量 pf，指向一个有两个 int 型参数的函数，该函数返回一个 double 型数据。

_____

三、程序填空题

1．以下程序的输出是_____。

```
#include <stdio.h>
void t(int x,int y,int cp,int dp)
{
    cp=x*x+y*y;
    dp=x*x-y*y;
}
int main( )
{
    int a=4,b=3,c=5,d=6;
    t(a,b,c,d);
    printf("%d %d\n",c,d);
    return 0;
}
```

2．有以下程序：

```
#include <stdio.h>
void num( )
{
    extern int x,y;
    int a=15,b=13;
    x=a-b;
    y=a+b;
}
int x,y;
int main( )
{
    int a=8,b=5;
    x=a+b;
    y=a-b;
    num( );
    printf("%d, %d\n",x,y);
    return 0;
}
```

运行程序后的结果为_____。

3．有以下程序：

```
#include <stdio.h>
```

```
void f1( );
int main( )
{
    int i=0;
    printf("%d",i);
    f1( );
    f1( );
    printf("%d",i);
    return 0;
}
void f1( )
{
    static int i=1;
    i++;
    printf("%d",i);
}
```

运行后的输出结果是_____。

4．有以下程序：

```
#include <stdio.h>
void f(int v,int w)
{
    int t;
    t=v;
    v=w;
    w=t;
}
int main( )
{
    int x=1,y=3,z=2;
    if (x>y)
        f(x,y);
    else if (y>z)
        f(y,z);
    else
        f(x,z);
    printf("%d, %d, %d\n",x,y,z);
    return 0;
}
```

运行后的输出结果是_____。

5．以下函数的功能是计算 sum=1+$\frac{1}{2!}+\frac{1}{3!}+\cdots+\frac{1}{n!}$，请填空完成程序。

```
_______ func(_______)
{
    double sum=0.0,f=1.0;
    int i;
    for (i=1;i<=n;i++)
    {
        f=f*_____;
        sum=sum+f;
    }
    return sum;
}
```

6．下列程序划横线处有误，要使得输出结果是“5,3”，应将划线部分改为_____。

```
#include <stdio.h>
void swap(int *a,int *b)
{
```

```
    int t;
    t=a; a=b; b=t;
}
int main( )
{
    int i=3,j=5,*p=&i,*q=&j;
    swap(p,q);
    printf("%d, %d\n",*p,*q);
    return 0;
}
```

四、编程题

1．编写函数，求整数 *x* 的 *n* 次幂。n 为正整数，x 和 n 在主函数中输入，计算结果在主函数中输出。

2．编写函数，将指定的字符打印 *n* 次。要输出的字符及打印次数 n 均在主函数中输入。

3．编写函数，判断输入的数是否为素数。要判断的数在主函数中输入，并在主函数中输出是否素数的信息。

4．编写函数，计算一个整数各位数字之和，如 123，各位之和为 1+2+3＝6。该整数及计算结果均在主函数中输入、输出。

5．编写函数，计算 5!。在主函数中调用该函数，并在主函数中输出计算结果。

6．在主函数中输入圆柱体的半径和高，编写函数计算出它的体积和表面积，并在主函数中输出计算结果。

7．编写函数，对主函数中的 3 个整数按从小到大的顺序排序，要求使用指针作为函数参数。在主函数中输入 3 个整数，并在主函数中输出排序后的结果。

第 4 章　简单构造数据类型

通过前几章的学习，已经能够编写较为复杂的程序了，但程序中处理的数据都是相对孤立的，每一个变量只能存储对应于该类型的一个数据。在实际应用中，数据往往是批量的，这些数据有相同的属性，或者有内在的联系。例如一个班的学生成绩、一个学生的信息（学号、姓名、联系电话、家庭住址等）等。用独立变量处理这样的数据，是很不方便的。如何将一些具有相同属性或保留内在联系的数据集合在一起，就需要构造数据类型。C 语言的构造数据类型有数组、结构体和共用体等。

本章学习数组的定义和使用、数组程序的编写、用数组处理字符串、用指针访问数组的方法等。

4.1　一 维 数 组

4.1.1　一维数组的引出

【例 4.1】 已知一个班 30 名学生参加了 C 语言考试，编写程序，输入每个学生的成绩，计算平均成绩，并输出每个学生的考试成绩和平均成绩。

分析：30 名学生的成绩对应着 30 个整数或实数，定义 30 个变量显然很不方便。分析一下，这 30 个数据都是描述学生的成绩信息，数据类型是一致的，都是整数或实数。对于相同类型的一组数，在 C 语言中可以用一维数组表示和存储。

30 名学生的考试成绩假设都是整数，可以定义一个一维数组“int a[30];”，表示定义了一个包含 30 个元素的数组，分别为：a[0],a[1],…,a[29]。每个数组元素相当于一个整型变量，分别存放每个学生的考试成绩。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int a[30];                  // 定义包含 30 个元素的数组 a，每个数组元素存放一个学生的成绩
    double sum=0,aver;
    int i;
    for (i=0;i<30;i++)
    {
        scanf("%d",&a[i]);                  // 输入每名学生的成绩
        sum=sum+a[i];                       // 成绩累加
    }
    aver=sum/30;                            // 求平均成绩
    for (i=0;i<30;i++)
        printf("%-4d",a[i]);                // 将所有学生的成绩输出
    printf("aver=%.2f\n",aver);             // 输出平均成绩
    return 0;
}
```

由上面的程序可以看出：数组能一次定义多个同类型的变量，可以方便地使用循环语句实现对每个元素的使用，数据的访问只需改变其下标即可。可见，数组的使用大大方便了程序设计。

小结：数组是为了处理一批类型相同的数据而引入的构造数据类型，是由类型相同、个数固定的元素组成的集合。数组中的所有元素都属于同一个数据类型。每个数组元素都是一个变量，其类型为

数组的类型，与相同类型的普通变量用法完全一样。

4.1.2　一维数组的定义和引用

1．一维数组的定义

一维数组的定义形式如下：

```
类型标识符 数组名[常量表达式];
```

例如：

```
int a[10];
```

它表示定义了一个名为 a 的整型数组，该数组共有 10 个元素，即 10 个整型变量，可以存放 10 个整数。

说明：

① 类型标识符表示数组各元素的数据类型，数组中的元素属于同一种数据类型。

② 数组名是用户自定义的标识符，遵循标识符的命名规则。数组名表示数组所占内存区域的首地址（即第 1 个数组元素 a[0]的地址）。

③ 常量表达式用方括号括起来，表示数组的大小，即数组元素的个数。

数组元素在数组中的序号称为下标。可以通过数组名和下标来唯一地确定每一个元素。数组可以具有多组下标，数组下标的组数称为数组的维数。只有一组下标的数组称为一维数组。

C 语言规定，数组的下标从 0 开始。如果定义了一个具有 *n* 个元素的数组，则其元素对应的下标为 0～*n*−1。

定义数组下标的常量表达式可以是整型常量或符号常量，例如：

```
#define M 40
char s[M];
int b[100];
```

以上定义都是合法的。尤其要注意，不能用变量来定义数组的大小。以下两种定义形式都是错误的：

```
int n=40;
int a[n];
```

或

```
int n;
scanf("%d",&n);
int a[n];
```

一维数组的元素被分配在一段连续的内存单元中，按下标顺序存放。例如，定义“int x[10];”，数组元素在内存的存放情况如图 4.1 所示。在 Visual C++ 6.0 集成环境中数组 x 所占的总内存大小为 4×10=40 个字节。

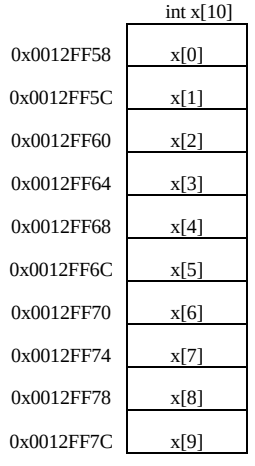

图 4.1　数组在内存中的存放

2．一维数组元素的引用

数组必须先定义后使用。定义完数组后，就可以使用数组元素了。C 语言规定，只能逐个引用数组元素而不能一次引用整个数组。数组元素的引用形式为：

```
数组名[下标]
```

引用数组元素时，下标可以是整型常量，也可以是已经赋值的整型变量或整型表达式。例如下列语句：

```
int a[5],i=3;
a[0]=1; a[1]=1; a[2]=2;
a[i]=a[i−1]+a[i−2]; a[2*2]=5;
```

执行程序段后，a[3]的值为 3，a[4]的值为 5。

注意：当数组长度为 *N* 时，数组的下标取值范围为 0,1,2,…,*N*1，如果引用元素 a[N]则是错误的，并没有 a[N]这个元素。例如，定义“int a[5];”，其元素范围是 a[0]～a[4]，如果出现语句“a[5]=12;”，将

造成下标越界。而 C 语言的编译系统不检查越界错误，系统会将 a[4]后的下一个存储单元赋值为 12，这样可能会破坏数组以外其他的数据！

对于数组元素的输入/输出操作可以用循环来实现。例如：

```
#include <stdio.h>
int main( )
{
    int i,a[10];
    for (i=0;i<=9;i++)              // 输入 10 个整数存入数组 a 中
        scanf("%d",&a[i]);
    for (i=9;i>=0;i--)
        printf("%4d",a[i]);         // 逆序输出这 10 个整数
    return 0;
}
```

3．一维数组的初始化

C 语言允许在定义数组时对各元素指定初值，称为数组的初始化。例如：

```
int a[10]={1,3,5,7,9,11,13,15,17,19};
```

初始化后，各数组元素的值分别为：a[0]=1，a[1]=3，a[2]=5，a[3]=7，a[4]=9，a[5]=11，a[6]=13，a[7]=15，a[8]=17，a[9]=19。

也可以只给部分数组元素赋初值，系统自动对其余元素赋一默认值。当元素为数值型数据时，默认值为 0，字符型数据时，默认值为'\0'，例如：

```
int a[10]={1, 3, 5, 7, 9};          // 等价于 int a[10]={1, 3, 5, 7, 9, 0, 0, 0, 0, 0};
```

注意：对于部分数组元素赋初值，在 Turbo C 2.0 编译环境中需要在定义前加 static 静态局部变量声明，如果不加，系统不会自动赋默认值，默认的元素值将不确定。而在 Visual C++ 6.0 编译环境中可不加，系统会自动赋默认值。

若对数组元素不赋初值，可在定义前加 static 静态局部变量声明，系统自动对所有元素赋默认值，例如：

```
static int a[5];                    // 等价于 static int a[5]={0, 0, 0, 0, 0};
```

对全部数组元素赋初值时，由于数据的个数已经确定，因此可以不指定数组长度，其长度由初值个数自动确定，例如：

```
int a[ ]={0, 1, 2, 3, 4};           // 等价于 int a[5]={0, 1, 2, 3, 4};
```

不允许指明的数组元素个数小于初值个数，例如：

```
int a[5]={0, 1, 2, 3, 4, 5};
```

在编译时出错。

注意：数组只有在初始化时可以整体赋值，使用时则不允许整体赋值。例如：

```
int a[ ]={1,2,3,4}, b[4];
b=a;                                // 该语句是错误的
```

4.1.3　一维数组程序举例

【例 4.2】用数组实现 Fibonacci 数列的前 20 项的存储。Fibonacci 数列如下：

$$\begin{cases} Fib_1=1 & (n=1) \\ Fib_2=1 & (n=2) \\ Fib_n=Fib_{n-1}+Fib_{n-2} & (n\geqslant 3) \end{cases}$$

分析：Fibonacci 数列的前两项均为 1，之后的每一项等于相邻前两项之和，即该数列为 1,1,2,3,5,8,13,21,…。可以用一维数组来存放 Fibonacci 数列各项的值，若 f[i]表示数列的第 i 项，它的前两项分别为 f[i–2]和 f[i–1]，所以 f[i]=f[i–2]+f[i–1]。

为了使项数与数组下标保持一致，即 f[0]不用，把第 1 项存放在 f[1]中，第 2 项存放在 f[2]中，依次类推。由于要用到 f[20]，所以数组大小至少应定义为 21。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int f[21]={0,1,1};                  // f[0]不用，随便赋一个初值即可，此处赋为 0
    int i;
    for (i=3;i<=20;i++)                 // 递推计算第 3–20 项
        f[i]=f[i–2]+f[i–1];
    for (i=1;i<=20;i++)
    {
        printf("%–12d",f[i]);
        if (i%5==0)                     // 控制每行输出 5 个数
            printf("\n");
    }
    printf("\n");
    return 0;
}
```

运行结果如下：

```
1           1           2           3           5
8           13          21          34          55
89          144         233         377         610
987         1597        2584        4181        6765
```

【例 4.3】 将 1～100 之内的素数打印出来。

分析：本题可用筛法先求出 1～100 之间的素数。据说筛法是古希腊的埃拉托斯特尼（Eratosthenes，约公元前 274～194 年）发明的，又称埃拉托斯特尼筛法（Sieve of Eratosthenes）。

首先把 1～100 存放在数组元素 a[1]～a[100]中，在计算过程中，把非素数的元素值置为 0，表示不是素数，最后输出非零的元素值，即所求的素数。求素数时，首先把 1 去掉（1 不是素数），令 a[1]=0。剩下的数中，循环以下操作：每次保留最小的数（第 1 次保留 2），该数是素数，然后去掉该数的全部倍数，它们都不是素数，把元素值置为 0，重复此过程，直到全部处理完毕，最后得到的就是 1～100 之间的全部素数。

算法步骤如下：

① 把整数 1～100 存放在数组元素 a[1]～a[100]中，并令 a[1]=1，a[2]=2，…。

② 将 1 去掉，因为 1 不是素数，令 a[1]=0 表示去掉。

③ 从剩下的数中取出值为非 0 的最小数 i（第 1 次 i=2），该数是素数，保留，元素值 a[i]不变。

④ 把 i 的所有倍数去掉，令相应的元素为 0 表示被去掉。

⑤ 重复③、④，直到全部处理完毕。

⑥ 输出所有值不为 0 的数，即是 1～100 之间的全部素数。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int i,j,n,a[101];                   // 要用到元素 a[100]，数组大小定义为 101
    for (i=1;i<=100;i++)                // 将 1～100 存入对应的数组元素中
        a[i]=i;
    a[1]=0;                             // 1 不是素数，去掉
    for (i=2;i<=100;i++)
    {
```

```
            if (a[i]==0)                    // 值为 0 表示已经去掉了，不需处理它的倍数
                continue;
            for (j=2*i;j<=100;j+=i)         // 去掉 a[i]的所有倍数
                if (a[j]!=0)                // 已被去掉的数不用处理
                    a[j]=0;                 // 该数不是素数，设置为 0 表示去掉
        }
        for (i=1,n=0;i<=100;i++)
        {
            if (a[i]!=0)                    // 不为 0 表示该数是素数
            {
                printf("%-4d",a[i]);
                n++;
                if (n%10==0)                // 控制每行输出 10 个数
                    printf("\n");
            }
        }
        printf("\n");
        return 0;
    }
```

运行结果如下：

```
2    3    5    7    11   13   17   19   23   29
31   37   41   43   47   53   59   61   67   71
73   79   83   89   97
```

4.2　二 维 数 组

4.2.1　二维数组的引出

【例 4.4】 假设考试共有 5 个科目，一个班有 20 名学生。输入所有学生的各科成绩，求出每名学生的总成绩。

分析：20 名学生的 1 个科目的成绩可以定义为一维数组 score1[20]，现在有 5 个科目的成绩，当然也可以定义 5 个一维数组，分别存放 20 名学生 5 个科目的成绩，但当考试科目再多时，显然这样定义是非常麻烦的。仔细分析可以发现，对于每名学生都有 5 个成绩，这些成绩都是整数，即都为相同类型的数据，可以把 5 个科目和 20 位学生看作一个二维表格，每名学生一行，每个科目一列。每一行就是一个一维数组。定义一个二维数组来存放表格中的数据。

20 名学生的 5 个科目成绩可定义为一个二维整型数组“int a[20][5];”，该数组含有 20×5 个数组元素，分别为：a[0][0], a[0][1], …, a[0][4], a[1][0], a[1][1], …, a[1][4], …, a[19][0], a[19][1], …, a[19][4]。在输入时，先输入第 1 个学生的第 1 门课成绩，再输入第 2 门课成绩，5 个科目的成绩都输入完后，再分别输入第 2 个学生的 5 个科目成绩，依次类推。因此，可以使用两层循环来控制，外层循环控制输入每名学生的成绩，内层循环控制输入该学生每门课程的分数。总成绩存放在一维数组 sum 中。

参考程序如下：

```
#include <stdio.h>
#define M 20                                // 定义符号常量 M，代表学生总人数
#define N 5                                 // 定义符号常量 N，代表 5 个考试科目
int main( )
{
    int a[M][N],sum[M];
    int i,j;
    for (i=0;i<M;i++)
    {
```

```
            sum[i]=0;                          // 第 i 个学生的总成绩清 0
            for (j=0;j<N;j++)                  // 依次输入第 i 个学生的成绩
            {
                scanf("%d",&a[i][j]);
                sum[i]=sum[i]+a[i][j];         // 计算第 i 个学生的总成绩
            }
        }
        for (i=0;i<M;i++)
        {
            printf("Stu%d:",i);
            for (j=0;j<N;j++)                  // 将每名学生的 5 个科目的成绩输出
                printf("%4d",a[i][j]);
            printf("%d\n",sum[i]);             // 输出第 i 个学生的总成绩
        }
        return 0;
    }
```

通过该例可以看出，有关一组相同类型的数据都可以用数组进行考虑。

4.2.2　二维数组的定义和引用

1．二维数组的定义

二维数组的定义形式如下：

类型标识符　数组名[常量表达式][常量表达式];

例如“int a[3][4];”，表示定义了一个 3 行 4 列的整型二维数组 a，含有 3×4=12 个数组元素。

根据数组的定义方式，可以将二维数组看作是一种特殊的一维数组，它的每一个元素（a[0]、a[1]、a[2]）又是一个一维数组，如图 4.2 所示。

a[0]→	a[0][0]	a[0][1]	a[0][2]	a[0][3]
a[1]→	a[1][0]	a[1][1]	a[1][2]	a[1][3]
a[2]→	a[2][0]	a[2][1]	a[2][2]	a[2][3]

图 4.2　二维数组

在内存中，二维数组的存放顺序是按行存储，即先顺序存放第 1 行的元素，再存放第 2 行的元素，如图 4.3 所示。

通常二维数组可以作为矩阵的形式进行考虑，第 1 维表示行，第 2 维表示列。

int a[3][4]

0x0012FF50	a[0][0]
0x0012FF54	a[0][1]
0x0012FF58	a[0][2]
0x0012FF5C	a[0][3]
0x0012FF60	a[1][0]
0x0012FF64	a[1][1]
0x0012FF68	a[1][2]
0x0012FF6C	a[1][3]
0x0012FF70	a[2][0]
0x0012FF74	a[2][1]
0x0012FF78	a[2][2]
0x0012FF7C	a[2][3]

图 4.3　二维数组在内存中的存放

2．二维数组的引用

二维数组元素的表示形式为：

数组名[下标][下标]

以下均是正确的二维数组引用：

a[1][2]　a[2−1][2*2−1]　a[i][j]　b[i][j]=a[j][i]

注意：二维数组的行下标和列下标均从 0 开始，引用时不能超过数组定义的范围。例如，在定义“int a[3][4];”中，二维数组 a 的最后一个数组元素为 a[2][3]，在使用中不能出现形如“a[3][4]=5;”的语句，因为不存在 a[3][4]这个元素。

3．二维数组的初始化

可以按以下形式对二维数组进行初始化。

① 按行给二维数组赋初值，例如：

```
int a[3][4]={{1,2,3,4}, {5,6,7,8}, {9,10,11,12}};
```

即把第 1 个花括号内的数据赋给第 1 行的各元素，第 2 个花括号内的数据赋给第 2 行的各元素……结果用矩阵表示为：

```
1    2    3    4
5    6    7    8
9    10   11   12
```

② 按数组存储顺序依次给各元素赋初值，例如：

```
int a[3][4]={1,2,3,4,5,6,7,8,9,10,11,12};
```

结果同方法①。

注意：此方法数据没有明显的界限，当数据较多时容易出错。

③ 可以对部分元素赋初值，其余元素自动赋默认值，例如：

```
int a[3][4]={{1}, {5}, {9}};
```

即只对各行第 1 列的元素赋初值，其余元素赋默认值 0，结果可表示为：

```
1    0    0    0
5    0    0    0
9    0    0    0
```

④ 只对数组各行中的某些元素赋初值，例如：

```
int a[3][4]={{1}, {0,5}, {0,0,9}};
```

则结果可表示为：

```
1    0    0    0
0    5    0    0
0    0    9    0
```

⑤ 只对数组中前面几行赋初值，后面各行的元素自动赋值为默认值，例如：

```
int a[3][4]={{1}, {3,5}};
```

则结果可表示为：

```
1    0    0    0
3    5    0    0
0    0    0    0
```

⑥ 当对全部数组元素赋初值时，第一维的长度可以不说明，但第二维的长度不能省略，例如：

```
int a[ ][3]={1,2,3,4,5,6,7,8,9};
```

即每行 3 个元素，共 3 行，结果可表示为：

```
1    2    3
4    5    6
7    8    9
```

⑦ 按行对部分元素赋初值，可以省略第一维的长度说明，例如：

```
int a[ ][3]={{0,0,3}, {0,2}, {1}};
```

则结果可表示为：

```
0    0    3
0    2    0
1    0    0
```

注意：同一维数组一样，不允许所提供的初值个数多于数组的元素个数。

4.2.3　二维数组程序举例

【例 4.5】 将一个二维数组的行和列元素互换，存到另一个二维数组中。例如：

$$a=\begin{bmatrix}1 & 2 & 3\\ 4 & 5 & 6\end{bmatrix} \qquad b=\begin{bmatrix}1 & 4\\ 2 & 5\\ 3 & 6\end{bmatrix}$$

分析：要将二维数组的行和列元素互换，即第 1 行元素变为第 1 列（a[0][0]变为 b[0][0]，a[0][1]变为 b[1][0]，a[0][2]变为 b[2][0]），第 2 行的元素变为第 2 列（a[1][0]变为 b[0][1]，a[1][1]变为 b[1][1]，a[1][2]变为 b[2][1]）……。观察发现，每一个元素都是从第 *i* 行第 *j* 列变为第 *j* 行第 *i* 列。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int a[2][3]={{1,2,3},{4,5,6}};
    int b[3][2],i,j;
    printf("array a:\n");
    for (i=0;i<2;i++)
    {
        for (j=0;j<3;j++)
        {
            printf("%4d",a[i][j]);
            b[j][i]=a[i][j];                    // 行列互换，实现转置
        }
        printf("\n");
    }
    printf("array b:\n");
    for (i=0;i<3;i++)
    {
        for (j=0;j<2;j++)
            printf("%4d",b[i][j]);
        printf("\n");
    }
    return 0;
}
```

运行结果如下：

```
array a:
   1   2   3
   4   5   6
array b:
   1   4
   2   5
   3   6
```

【例 4.6】 有一个 3×4 的矩阵，编程求出最大元素的值，以及其所在的行号和列号。

分析：找最大数的方法同以前一样，先设矩阵的第一个元素的值（a[0][0]）为最大值 max，并记下行号、列号，然后依次访问矩阵的每个元素，与 max 比较，若当前元素比 max 大，则修改 max，并记下该元素的行号和列号。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int i,j,row,colum,max;
    int a[3][4]={{1,2,3,4},{9,8,7,6},{-10,10,-5,2}};
    max=a[0][0]; row=0; column=0;           // 记录第 1 个数，及其行号、列号
    for (i=0;i<3;i++)
        for (j=0;j<4;j++)
            if (a[i][j]>max)
            {
                max=a[i][j];                // 修改最大数
                row=i;                      // 记录行号
                colum=j;                    // 记录列号
```

```
            }
        printf("max=%d,row=%d,colum=%d\n",max,row,colum);
        return 0;
    }
```

运行结果如下：

```
max=10,row=2,colum=1
```

4.3　字符数组与字符串

4.3.1　字符数组的引出

C 语言中，把字符串看成是由若干字符组成的字符序列。一个字符串用一个一维字符数组来存放，把字符串中的各字符依次存放在对应的数组元素中。在对字符串进行处理时，可以逐个字符进行处理，也可以对整个字符串进行处理。

【例 4.7】 预先设置一个 *N* 位长度的密码，由数字、字母、符号等组成。程序运行时要求用户输入密码，如果用户输入的密码正确，输出“Welcome!”；若密码输入错误，输出“Wrong!”。

方法一：采用逐个字符的方式处理。预设的密码可存放在长度为 *N* 的字符数组 code[*N*]中，用户输入的密码存放在 user[*N*]中。进行密码判断时，逐个比较对应的每一位字符，只要有 1 位不相同，则说明密码错误，全部字符都相同则说明密码正确。

参考程序如下：

```
#include <stdio.h>
#define N 10                            // 符号常量 N 表示预设密码的长度
int main( )
{
    char code[N]={'A','B','C','1','2','3','*','x','y','z'};  // 定义字符数组，存放预设的密码
    char user[N];                       // 字符数组 user 存放用户输入的密码
    int i;
    printf("Input password: ");         // 提示用户输入密码
    for (i=0;i<N;i++)                   // 逐个字符读入用户输入的 N 位密码
        user[i]=getchar( );
    for (i=0;i<N;i++)                   // 逐个字符判断密码是否相同
        if (user[i]!=code[i])
            break;
    if (i==N)                           // 说明没有不相同的字符，密码正确
        printf("Welcome!\n");
    else                                // 即 i<N，说明有不相同的字符，密码错误
        printf("Wrong!\n");
    return 0;
}
```

方法二：采用字符串的方式处理。预设密码以字符串的形式存放在字符数组 code[*N*]中，用户也以字符串的形式输入密码。进行密码判断时，采用字符串比较函数直接比较即可。

参考程序如下：

```
#include <stdio.h>
#include <string.h>                     // 使用字符串处理函数时需要包含的头文件
#define N 10
int main( )
{
    char code[N+1]="ABC123*xyz";  // 字符串末尾有结束标注'\0'
    char user[50];                      // 用户输入的密码可能较长
    printf("Input password: ");
```

```
    gets(user);                          // 以字符串的形式读入用户密码
    if (strcmp(code,user)==0)            // 比较两个字符串，函数值为 0 说明两个字符串相等
        printf("Welcome!\n");
    else                                 // 即函数值不为 0，说明两个字符串不相等
        printf("Wrong!\n");
    return 0;
}
```

小结：采用逐个字符的方式处理时，密码的位数必须预先规定，输入的密码位数只能是预先规定的 N 位，不能少于 N 位或多于 N 位，判断密码是否相同时也只能逐位比较。采用字符串的方式处理时，用户输入的密码位数可长可短，判断密码是否相同时，直接调用字符串处理函数即可，方便快捷。在使用字符串处理函数时，需要包含 string.h 头文件。

4.3.2　字符数组的定义和引用

1．字符数组的定义

用来存放字符型数据的数组称为字符数组。字符数组的每个元素只能存放 1 个字符。例如：

```
char c[5];
c[0]='C';   c[1]='h';   c[2]='i';   c[3]='n';   c[4]='a';
```

表示定义了一个可以存储 5 个字符的字符数组 c。赋值以后数组的状态如图 4.4 所示。

c[0]	c[1]	c[2]	c[3]	c[4]
C	h	i	n	a

图 4.4　字符数组存储情况

2．字符数组的初始化

在进行字符数组初始化时，可以逐个地将字符赋值给数组中的元素，例如：

```
char c[5]={'C', 'h', 'i', 'n', 'a'};
```

如果指定的初值个数小于数组长度，按顺序赋值后，其余元素自动赋空字符'\0'，例如：

```
char str[10]={'B', 'e', 'i', 'J', 'i', 'n', 'g'};
```

此时：str[7]=str[8]=str[9]='\0'。

若初值个数大于数组长度，按语法错误处理。

若提供的初值个数与预定的数组长度相同，在定义时可省略数组长度，例如：

```
char c[ ]={'C', 'h', 'i', 'n', 'a'};   // 省略的长度应为 5
```

也可以定义和初始化一个二维字符数组，例如：

```
char diamond[5][5]={{' ', ' ', '*'},{' ', '*', ' ', '*'},{'*', ' ', ' ', ' ', '*'},{' ', '*', ' ', '*'},{' ', ' ', '*'}};
```

这些字符排列出来如图 4.5 所示。

```
  *
 * *
*   *
 * *
  *
```

图 4.5　字符排列

3．字符数组的引用

可以引用字符数组中的一个元素，得到一个字符。例如：

```
// 输出一个字符串
#include <stdio.h>
int main( )
{
    char c[14]={'I', ' ', 'a', 'm', ' ', 'a', ' ', 's', 't', 'u', 'd', 'e', 'n', 't'};
    int i;
    for (i=0;i<14;i++)
       printf("%c",c[i]);
    printf("\n");
}
```

运行结果为：

```
I am a student
```

4.3.3　字符串的使用

1．字符串的定义

字符串是由双引号括起来的字符序列，例如：

```
"Hello World!"　　"China"　　"How are you?"
```

在实际问题中经常会遇到使用一串字符的情况，如学生姓名、英文单词、一串密码等。C 语言中没有字符串变量，字符串存放在字符数组中。

字符常量只有一个字符，字符串没有固定长度。空格是合法的字符，不能作为结束标志，C 语言规定用空操作符'\0'作为字符串的结束标志。程序在定义时会在每个字符串的后面自动加上一个空操作符'\0'以示区别，在计算字符串长度时，'\0'不计入字符串的长度。

字符串必须以'\0'作为结束标志，它只表示一个字符串的结束，没有任何具体含义。在存储字符串时，虽然'\0'不会计入字符串长度中，但'\0'将会占用 1 个元素的存储空间，所以在定义字符数组时，应在字符串长度的基础上增加 1 个元素。

例如，字符串"How are you?"有 12 个字符，但在定义时字符数组长度至少应为 13，在内存中实际占 13 个存储空间，包含一个字符串结束标志'\0'。

有了'\0'作为字符串结束标志后，字符串就可以作为一个整体在程序中使用了。

2．字符串的初始化

可以用字符串整体对字符数组进行初始化，例如：

```
char c[6]={"China"};
```

也可以写成：

```
char c[ ]="China";
```

也等价于：

```
char c[6]={'C', 'h', 'i', 'n', 'a', '\0'};
```

但不能写成：

```
char c[5]="China";
```

注意两种赋值方式的差别。请思考，“char c[5]={'C', 'h', 'i', 'n', 'a'};”是否合法？

注意：字符数组并不要求包含'\0'，这一点与字符串不同。

3．字符串的输入与输出

字符串的输入与输出可以有以下几种方法。

（1）用"%c"格式控制符实现逐个字符的输入/输出

例如：

```
#include <stdio.h>
int main( )
{
    int i=0;
    char str[20];                  // 定义字符数组 str，最多可存放 20 个字符
    str[i]=getchar( );             // 输入第 1 个字符
    while (str[i]!='\n'&&i<19)     // 输入的字符不是'\n'且不超过 19 个
    {
        i++;
        str[i]=getchar( );         // 使用 getchar 函数不断读入下一个字符
    }
    str[i]='\0';                   // 在字符串末尾赋结束标志'\0'
```

```
    for (i=0;str[i]!='\0';i++)        // 逐个字符输出，直到'\0'时结束输出
        printf("%c",str[i]);
    return 0;
}
```

（2）用"%s"格式控制符对数组进行整体输入和输出

例如：

```
#include <stdio.h>
int main( )
{
    char str[20];
    scanf("%s",str);
    printf("%s\n",str);
    return 0;
}
```

注意：

① 从键盘输入字符串时不需加双引号""。

② 用 scanf 输入字符串时，空格和回车符都会作为字符串的分隔符，即 scanf 的%s 格式控制符不能用来输入包含空格的字符串。

③ 用 printf 函数的%s 格式输出字符串时，输出项是字符数组名，而不是数组元素名。

④ 用 scanf 函数的%s 格式输入字符串时，输入项是字符数组名，既不是数组元素，也不能在数组名前加取地址符号'&'。

⑤ 如果数组长度大于字符个数，只输出到字符串结束标志'\0'为止。例如：

```
char c[10]="China";
printf("%s",c);
```

则输出为 China。

（3）用 gets 和 puts 函数实现字符串的输入和输出

例如：

```
#include <stdio.h>
int main( )
{
    char str[20];
    gets(str);
    puts(str);
    return 0;
}
```

注意：输入有空格的字符串时应使用 gets 函数，它可以读入包括空格在内的全部字符直到遇到回车符为止。用 gets 函数输入字符串时，若输入字符数大于字符数组的长度，则多出的字符会存放在数组的合法存储空间之外。puts 函数一次输出一个字符串，输出时将'\0'自动转换成换行符。

说明：

① 输出字符不包括结束符'\0'。

② 注意输入/输出格式，例如：

```
char str[20],ch;                    char str[20],ch;
scanf("%s",str);                    scanf("%c",&ch);
printf("%s",str);                   printf("%c",str[0]);
```

③ 如果一个字符数组中包含多个'\0'，则遇第一个'\0'时输出结束。例如：

```
char str[10]={'a', 'b', 'c', '\0', 'd', 'e', '\0'};
printf("%s",str);
```

输出结果为：

```
abc
```

4．多个字符串的存储

实际应用中，经常需要存储多个字符串。例如，存储 30 个学生的姓名，就需要 30 个字符数组，定义及操作都很不方便。此时，可以定义一个有 30 行的字符型二维数组，二维数组的每一行用于存放一个字符串，列数表示存放字符的最大个数。例如：

```
char str[30][40];
```

它表示数组 str 可以存储 30 个字符串，每个字符串最多可以包含 39 个字符。注意，每个字符串末尾还有一个字符串结束标志'\0'。

由于二维数组元素在内存中是按行连续分配存储空间的，因此字符型二维数组在内存中是按字符串的顺序来存储的，即先存储完第 1 个字符串，再存储第 2 个字符串，直到所有的字符串都存储完。

5．字符串处理函数

由于字符串的特殊性，很多操作都不能用处理数值型数据的方法来完成，如赋值、比较等。此外，字符串还有一些特殊的操作，如计算字符串长度、查找字符串的子串、字符串的连接等。

C 语言编译系统为字符串提供了一系列字符串处理函数来完成这些工作，这些函数都包含在头文件 string.h 中，在使用这些函数时需要包含该文件。

（1）字符串复制函数 strcpy

strcpy 函数的格式为：

```
strcpy(字符数组 1,字符串 2)
```

其功能是，将字符串 2 复制到字符数组 1 中。

例如：

```
char str1[10], str2[ ]="China";
strcpy(str1,str2);
```

执行后，str1 的状态如图 4.6 所示。

C	h	i	n	a	\0	\0	\0	\0	\0

图 4.6　字符数组 str1 的存储情况

说明：

① “字符数组 1”的长度不应小于“字符串 2”的长度。

② “字符数组 1”必须写成数组名形式或字符型指针变量。“字符串 2”可以是字符数组名、字符型指针变量或字符串常量。

③ 复制时连同'\0'一起复制。

④ 不能用赋值语句将一个字符串常量或字符数组直接赋给另一个字符数组。例如，下面的赋值语句都是不合法的：

```
str1="China";
str2=str1;
```

字符串复制函数的扩展格式为：

```
strncpy(字符数组 1,字符串 2,n)
```

其作用是，将字符串 2 的前 *n* 个字符复制到字符数组 1 中。

例如：

```
char c1[10], c2[ ]="abcdef";
strncpy(c1,c2,3);
```

执行后，c1 的状态如图 4.7 所示。

a	b	c	\0	\0	\0	\0	\0	\0	\0

图 4.7　字符数组 c1 的存储情况

（2）字符串连接函数 strcat

strcat 函数的格式为：

```
strcat(字符数组 1,字符数组 2)
```

其功能是，连接两个字符数组中的字符串，把字符串 2 连接到字符串 1 的后面，结果仍存放在“字符数组 1”中。

例如：

```
char str1[20]="China ",str2[ ]="Bei jing";
printf("%s",strcat(str1,str2));
```

输出结果为：

```
China Bei jing
```

连接前后的状况如图 4.8 所示。

str1:	C	h	i	n	a		\0	\0	\0	\0	\0	\0	\0	\0	\0
str2:	B	e	i		j	i	n	g	\0						
连接后的 str1:	C	h	i	n	a		B	e	i		j	i	n	g	\0

图 4.8　两个字符串连接前后的存储情况

说明：

① 字符数组 1 必须定义得足够大，以便容纳连接后的新字符串。

② 连接前两个字符串后面都有一个 '\0' ，连接时将字符串 1 后面的'\0'去掉，只在新串最后保留一个'\0'。

（3）字符串比较函数 strcmp

strcmp 函数的格式为：

```
int strcmp(字符串 1,字符串 2)
```

其功能是，比较“字符串 1”和“字符串 2”（从左到右逐个字符比较 ASCII 值的大小，直到出现的字符不一样或遇到'\0'为止，比较结果由函数返回）。

① 若“字符串 1”=“字符串 2”，函数的返回值为 0。

② 若“字符串 1”>“字符串 2”，函数的返回值为一个正整数。

③ 若“字符串 1”<“字符串 2”，函数的返回值为一个负整数。

例如：

```
strcmp("China","China")      strcmp("35+78","4")
strcmp(str,"China")          strcmp("computer","compare")
```

注意：对两个字符串是否相同的比较，不能用以下形式：

```
if (str1==str2) printf("yes");
```

而只能用：

```
if (strcmp(str1,str2)==0) printf("yes");
```

（4）测试字符串长度函数 strlen

strlen 函数的格式为：

```
int strlen(字符串)
```

其功能是，测试字符串长度（函数的值为字符串的实际长度，不包括'\0'在内）。

例如：

```
char str[20]="China";
printf("%d\n",strlen(str));
```

输出结果不是 20，也不是 6，而是 5。

也可以直接测字符串常量的长度，例如：

strlen("China")

（5）大小写转换函数 strupr 和 strlwr

大小写转换函数的格式分别为：

strupr(字符串)　　strlwr(字符串)

strupr 函数的功能是将字符串中的小写字母转换为大写字母。strlwr 函数的功能是将字符串中的大写字母转换为小写字母。例如：

strupr("abC")=="ABC" strlwr("abC")=="abc"

【例 4.8】 输入 3 个字符串，找出最大的字符串。

分析：3 个字符串可考虑用二维字符数组来存放，设一个二维的字符数组 str，大小为 3×20，即有 3 行 20 列，每一行可以容纳 20 个字符，如图 4.9 所示。

str[0]:	C	h	i	n	a	\0	\0	\0	…	\0	\0
str[1]:	A	m	e	r	i	c	a	\0	…	\0	\0
str[2]:	T	u	r	k	e	y	\0	\0	…	\0	\0

图 4.9　二维数组 str[3][20]

可以把 str[0]、str[1]、str[2]看作 3 个一维字符数组，它们各有 20 个元素。可以把它们如同一维数组那样进行处理。可以用 gets 函数分别读入 3 个字符串。经过两次比较，可得值最大者，并把它放在一维字符数组 string 中。

参考程序如下：

```
#include <stdio.h>
#include <string.h>
int main( )
{
    char string[20];
    char str[3][20];
    int i;
    for (i=0;i<3;i++)
        gets(str[i]);
    strcpy(string,str[0]);                    // 假设第 1 个字符串最大
    for (i=1;i<3;i++)                         // 依次与每个字符串进行比较
        if (strcmp(str[i],string)>0)
            strcpy(string,str[i]);
    printf("largest string:\n%s\n",string);
    return 0;
}
```

4.3.4　字符数组程序举例

【例 4.9】 输入一行字符，统计其中有多少个单词，已知单词之间用空格分隔开。

分析：假设单词之间仅由空格分隔，则单词的数目可以由空格出现的次数决定（连续的若干个空格作为出现一次空格，一行开头的空格不统计在内）。如果测出某一个字符为非空格，而它前面的字符是空格，则表示“新的单词开始了”，此时 num（单词数）累加 1。如果当前字符为非空格而其前面的字符也是非空格，则意味着仍然是原来那个单词的继续，num 不应再累加 1。定义一个变量 word，用来表示前一个字符是否空格，若 word 等于 0，则表示前一个字符是空格；如果 word 等于 1，意味着前一个字符为非空格。因此在程序中，判断当前字符是否空格，然后置 word 的值，如图 4.10 所示，其 N-S 图如图 4.11 所示。

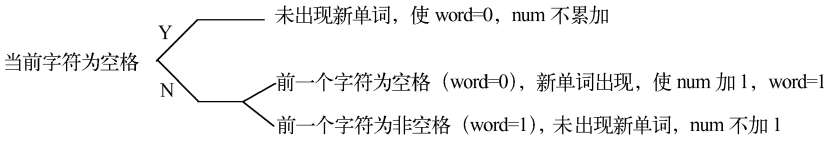

图 4.10　统计单词

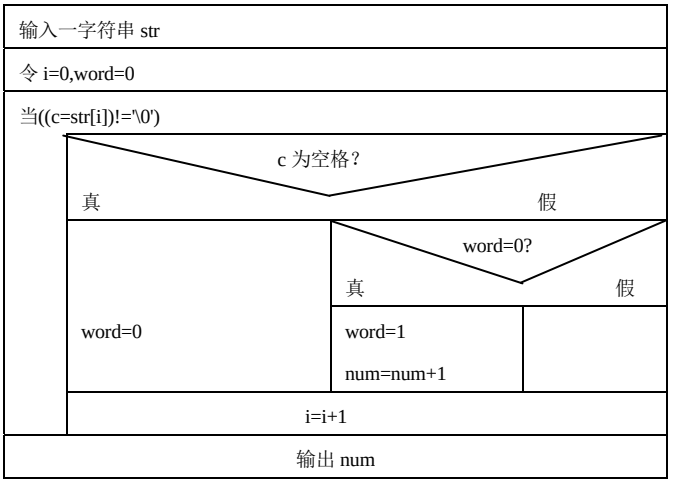

图 4.11　N-S 图

参考程序如下：

```
#include <stdio.h>
int main( )
{
    char str[100];
    int i,num=0,word=0;          // num 用来统计单词个数，word 作为判别是否单词的标志
                                 // 初值设为 0，表示前面是空格
                                 // 若 word=0 表示未出现新单词，如出现新单词 word 就置成 1
    char c;
    printf("Input a statement:\n");
    gets(str);
    for (i=0;(c=str[i])!='\0';i++)
        if (c==' ')              // 当前字符为空格，word 置为 0
            word=0;
        else if (word==0)        // 当前字符为非空格，且前一个字符为空格，新单词开始
        {
            word=1;
            num++;               // 单词数增 1
        }
    printf("There are %d words in the line.\n",num);
    return 0;
}
```

运行结果如下：

```
Input words:
I am a boy↙
There are 4 words in the line.
```

【例 4.10】 编写一个程序，将两个字符串连接起来。要求不使用 strcat 函数。

分析：两个字符串进行连接时，将第一个字符串的结束标志去掉，第二个字符串的第一个字符连接到第一个字符串的结束标志的位置。注意，连接后的字符串必须有串结束标志。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    char s1[80],s2[40];
    int i=0,j=0;
    printf("Input string1:");
    gets(s1);
    printf("Input string2:");
    gets(s2);
    while (s1[i]!='\0')              // 查找 s1 中'\0'所在的位置
        i++;
    while (s2[j]!='\0')              // 从 s1 中'\0'所在位置开始复制 s2
        s1[i++]=s2[j++];
    s1[i]='\0';                      // 注意：s2 中的'\0'并没有复制过去，必须单独赋值
    printf("The new string is: %s\n",s1);
    return 0;
}
```

运行结果如下：

```
Input string1:country↙
Input string2:side↙
The new string is:countryside
```

4.4　数组与指针

前面已经介绍，指针变量可以指向同类型的普通变量。数组即一组变量的集合，一个数组元素相当于同类型的普通变量，那么指针变量能否指向数组元素或是数组呢？当然可以！指针变量指向数组元素的含义是，指针变量存储某一数组元素的地址。而对于数组来说，数组名表示数组在内存中的首地址，知道了首地址即可依次取出该数组中的每一个数组元素，因此可以定义一个指针变量指向数组的第一个元素，后面的数组元素可以通过指针的移动来获得。下面来学习数组与指针的关系。

4.4.1　一维数组与指针

数组名代表数组在内存中的首地址，表示的是地址信息，因此数组名也可以理解为一个指针，不过数组名是一个指针常量，不能改变。可以用数组名将数组的首地址赋给一个指针变量，找到了数组的首地址，就相当于找到了数组中的每一个元素。

1．指向数组的指针

定义指向数组的指针就是将数组的首地址赋值给指针变量。要注意，指针变量的基类型要与数组元素的类型一致。例如：

```
int a[10];          // 定义整型数组
int *p;             // 定义整型指针变量
p=&a[0];            // 指针变量 p 指向 a[0]
```

由于数组名就代表数组在内存中存储的首地址，即第 1 个元素的地址，因此，“p=&a[0];”等价于“p=a”。

那么如何让 p 指向下一个元素 a[1]呢？有必要介绍一下指针变量加法与其他加法的区别。

指针变量中存储的是地址，因此，指针变量和整数进行加、减运算是按地址的运算规则进行的。当一个指针变量和整数进行加、减运算后，实际上会产生一个新的地址值，新地址值和加减运算时的

整数值及指针变量的基类型有关。

例如，一个指针变量 p 和一个整数 n 执行了如下形式的运算：

```
p=p+n;
p=p−n;
```

则指针变量 p 的值按以下公式计算：

$$p\text{ 的新值}=p\text{ 的旧值}\pm n\times \text{sizeof(指针基类型)}$$

因此，从整体上看，当一个指针变量加减 n 后，实际上是把指针变量的指向向后或向前移动了 n 个基类型单元。

所以，若有“p=&a[0];”则执行“p=p+1;”后，p 就指向了 a[1]。同理，使用该方法可使指针变量向后或向前移动，指向其他元素，因此能够方便地利用指针访问数组各元素。

2．通过指针引用数组元素

前面介绍的对数组元素的引用都是采用了下标法，如 a[i]。由于指针可以指向数组，因此引用数组元素时也可以通过指针变量来访问。

当指针指向一个数组元素时，可以通过改变指针变量中的地址值，即改变指针的指向使指针指向其他数组元素。使用指针法能使目标程序质量高（占内存少、运行速度快），就像是一根指针在内存中移动，以此来存取数组元素的值。

如果指针变量 p 已指向数组中的一个元素。C 语言中用 p+1 表示让指针指向下一个数组元素，它不是指简单的地址值加 1，其关系如图 4.12 所示。

由图 4.12 可以看出，p+i 表示 a[i]的地址，则*(p+i)表示数组元素 a[i]。

可以用多种方法访问一维数组各元素，例如：

```
#include <stdio.h>
int main( )
{
    int a[5]={1, 3, 5, 7, 9};
    int i,*p=a;
    for (i=0;i<5;i++) printf("%d ",a[i]);
    printf("\n");
    for (i=0;i<5;i++) printf("%d ",*(a+i));
    printf("\n");
    for (i=0;i<5;i++) printf("%d",p[i]);
    printf("\n");
    for (i=0;i<5;i++) printf("%d",*(p+i));
    printf("\n");
    for (;p<a+5;p++) printf("%d",*p);
    printf("\n");
    p=a;          // 此语句必不可少，p 重新指向 a[0]
    while (p<a+5) printf("%d",*p++);
    printf("\n");
    return 0;
}
```

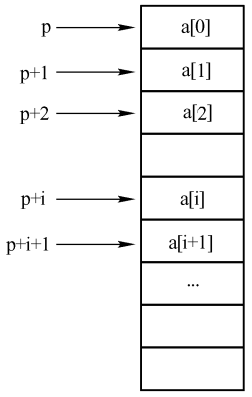

图 4.12　指针与数组

在使用指针变量时，要注意以下几点：

① 可以改变指针变量的值，但指针常量不可改变。例如，p++合法，但 a++不合法（因为 a 是数组名，是指针常量，而 p 是指针变量）。

② 要注意指针变量的当前值，指向哪一个数组元素。

③ 使用指针变量指向数组元素时，应保证指向数组中有效的元素。

④ 注意指针变量的运算（指向运算符的优先级最高）：

● p++：使 p 指向下一个元素；

- *p++：等价于*(p++)，先获得 p 当前指向的元素值，然后 p 值加 1，指向下一个元素；
- *(++p)：先使 p 指向下一个元素，再获得该元素的值；
- (*p)++：使 p 所指向的元素值加 1，p 的值没有改变，仍指向原元素。

【例 4.11】 一个数组存储了 10 名学生的成绩。输出所有学生的成绩，以及最高分和最低分。

分析：本题可用一般的方法来访问数组元素，也可以用指向一维数组元素的指针来访问数组元素。用指针时，首先将定义的指针变量指向数组的第一个元素，然后通过指针指向的改变来访问不同的数组元素，从而输出成绩和求取最大值、最小值。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int score[10],*p=&score[0];          // 指针变量 p 指向 score[0]
    int i,max,min;                       // max、min 分别存放成绩的最大值和最小值
    for (i=0;i<10;i++)
        scanf("%d",p++);                 // 读入各数组元素，并使指针值自增
    p=score;                             // 指针变量 p 重新指向第一个元素
    max=*p;                              // 以 score[0]作为最大值的初值
    min=*p;                              // 以 score[0]作为最小值的初值
    for (;p<score+10;p++)                // 依次访问各数组元素
    {
        if (*p>max) max=*p;
        if (*p<min) min=*p;
    }
    for (p=score;p<score+10;p++)
        printf("%–4d",*p);               // 利用指针变量输出数组元素
    printf("max=%4d,min=%4d\n",max,min);
    return 0;
}
```

注意：在上面的程序中，利用指针输入数组元素的值，每输完一个值指针就指向下一个数组元素，当输入结束时，指针 p 已不再指向 score 数组中的任何一个元素了。可是后面还要用指针 p 参与比较数组元素的大小，所以在执行完输入后，指针 p 要重新赋值，指向数组的第一个元素 score[0]。

4.4.2　多维数组与指针

在内存中，多维数组的元素也是按下标顺序存放的。一个指针变量可以指向一维数组中的元素，也可以指向多维数组中的元素。

学习本小节概念时，请熟记下面两组等价式：

x[i] 等价于 *(x+i)　　&x[i] 等价于 x+i

1．多维数组元素的地址

下面先回顾多维数组的性质，以二维数组为例。

设一个数组定义为：

```
int a[3][4];
```

其中，a 表示数组在内存中的首地址，也就是数组中第 1 个元素（也是第一行）的首地址，它是一个指针常量，其值由系统在编译时确定，程序运行期间不能改变。

该二维数组可以理解为，它是一个一维数组，含有 3 个元素，每个元素又是一个一维数组，该一维数组含有 4 个元素，每个元素都是 int 类型。

思考：a+1 代表哪个元素的地址？a[0]、a[1]、a[2]分别代表什么？

二维数组的逻辑结构图如图 4.13 所示。其中，a[i]等价于*(a+i)，a[i][j]等价于*(*(a+i)+j)。

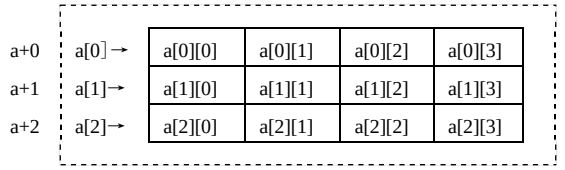

图 4.13　二维数组的逻辑结构图

2．指向多维数组元素的指针变量

指向由 *m* 个元素组成的一维数组的指针变量的定义为：

```
类型名 (*指针变量名)[长度];
```

例如：

```
int (*pa)[4];
```

表示 pa 是指针变量，它指向一个 int 型一维数组，该数组包含 4 个元素。

说明：该定义与定义“int *pa;”及“int *pa[4];”的含义不同。

思考：如果执行 pa++，则 pa 实际增加了多少？

4.4.3　数组作为函数参数

第 3 章讲述了函数的定义与使用，知道了函数之间主要是通过调用来联系的。实际上，函数之间的调用是通过参数之间的值传递或地址传递来实现的。下面来看数组与函数之间的关系。

对于单个的数组元素，其实质就是一个普通变量，可以作为函数的参数。数组元素作为函数参数时，也是单向值传递，即参数之间传递的是变量的值，这种值的传递是单向的。例如，在 main 函数中有以下调用：

```
ave=fun(a[0],a[1],a[2]);              // 在主函数中调用函数 fun
…
float fun(float a,float b,float c)    // fun 函数
{
    float sum,aver;
    sum=a+b+c;
    aver=sum/3.0;
    return(aver);
}
```

在调用 fun 函数时，将 a[0]、a[1]和 a[2]的值分别传送给形参 a、b 和 c，在求出平均值 aver 后，将 aver 的值返回到 main 函数中，赋值给变量 ave。

1．数组名作为函数参数

在程序设计过程中，经常需要在另外的函数中处理整个数组。如例 4.1，可以编写一个函数专门计算学生成绩的平均值。既然数组是一个整体，在内存中是连续存放的，而数组名就表示了该数组的首地址，相当于一个指针常量。因此，将数组名作为参数传递，把数组首地址传递到被调函数中，在被调函数中，知道了数组的首地址，就能够利用该地址访问数组中的所有元素。

当用数组名做函数参数时，实参传给形参的是一个地址值，此时，形参必须定义成指针变量的形式，以接收从实参传过来的地址，然后就可以通过形参指针变量来访问对应数组中的元素。因此，函数形参可以定义成下面两种形式：

```
void f(int arr[N])        // 下标法
void f(int *arr)          // 指针法
```

由于在被调函数中所声明的形参数组的大小实际上是不起任何作用的（形参数组的标识符实际只是一个地址），因此形参数组定义时可以不指定长度，一般另外设置一个参数传递数组的长度。函数的定义形式可改写为：

```
void f(int arr[ ],int n)
```

在主调函数中，对应的函数调用语句应为：

```
f(a,n);                    // a 为实参数组名，n 为数组的长度
```

用数组名做实参进行参数传递时，系统并不给形参数组分配内存空间，它将实参数组的首地址赋给形参，这样形参也就指向了内存中的同一地址。也就是说，对应的实参、形参都指向相同的内存空间。因此，在被调函数中对形参数组进行操作，实际上就是对实参数组进行操作。如果在被调函数中改变形参数组中元素的值，当然也就改变了实参数组元素的值（这一点也与一般变量作为函数参数不同）。

归纳起来，如果有一个实参数组，想在函数中改变此数组的元素的值，实参与形参的对应关系有以下 4 种。

① 形参与实参都用数组名

例如：

```
#include <stdio.h>
f(int x[ ],int n)
{
    …
}
int main( )
{
    int a[10];
    …
    f(a,10);
    …
}
```

② 实参用数组名，形参用指针变量

例如：

```
#include <stdio.h>
f(int *x,int n)
{
    …
}
int main( )
{
    int a[10];
    …
    f(a,10);
    …
}
```

③ 实参与形参都用指针变量

例如：

```
#include <stdio.h>
f(int *x,int n)
{
    …
}
int main( )
{
    int a[10],*p;
    p=a;
    …
    f(p,10);
    …
}
```

④ 实参为指针变量，形参为数组名

例如：

```
#include <stdio.h>
f(int x[ ],int n)
{
    …
}
int main( )
{
    int a[10],*p;
    p=a;
    …
    f(p,10);
    …
}
```

注意：实参数组名是指针常量，形参数组名是指针变量。

【例 4.12】 将数组 a 中的 n 个数逆序存放。

分析：所谓逆序存放，是指数组前后的对应元素相互交换。即元素 a[0]与 a[n−1]交换（假设数组元素存放在 a[0]～a[n−1]中），元素 a[1]与 a[n−2]交换，……，元素 a[i]与 a[n−1−i]交换。需要注意的是，只需循环到 n/2−1 即可。

参考程序如下：

```
#include <stdio.h>
void inv(int x[ ],int n);
int main( )
{
    int a[10]={3,7,9,11,0,6,7,5,4,2};
    int i;
    printf("the original array:\n");
    for (i=0;i<10;i++) printf("%d ",a[i]);
    inv(a,10);
    printf("\nthe array has been inverted:\n");
    for (i=0;i<10;i++) printf("%d ",a[i]);
    printf("\n");
    return 0;
}
void inv(int x[ ],int n)
{
    int temp,i;
    for (i=0;i<n/2;i++)
    {
        temp=x[i]; x[i]=x[n−i−1]; x[n−i−1]=temp;
    }
}
```

实参、形参数组在内存中的存储情况如图 4.14 所示，其中，x 相当数组 a 的别名，实际是同一个数组，因为它们有相同的首地址，占用同一段内存空间。

函数 inv 可改写为：

```
void inv(int *x,int n)
{
    int *p,*q,temp;
    p=x; q=x+n−1;
    for (;p<q;p++,q−−)
    {
        temp=*p; *p=*q; *q=temp;
    }
}
```

形参、实参在内存中的访问情况如图 4.15 所示。

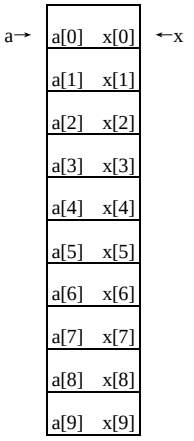

图 4.14　实参、形参数组在内存中的存储情况

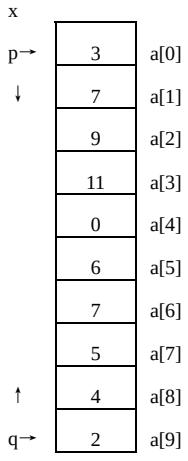

图 4.15　实参、形参在内存中的访问情况

2．多维数组作为函数参数

多维数组也可以作为函数的参数。当多维数组元素作为函数实参时，形参需定义为同类型的变量；当多维数组名作为函数实参时，形参需定义为多维数组。定义形参数组时可以指定每一维的大小，也可省略第一维的大小说明（但不能省略第二维及其他高维的大小说明）。例如：

```
int fun(int array[3][4])　　或　　int fun(int array[ ][4])
```

但不能写成：

```
int fun(int array[ ][ ])
```

也不能写成：

```
int fun(int array[3][ ])
```

实参数组可以大于形参数组，此时实参中只有前面的数据起作用。

【例 4.13】 编写一个函数，求 3×4 矩阵中的最大元素并返回该值。

分析：先使变量 max 的初值为矩阵中第一个元素的值，然后依次将矩阵中各个元素与 max 相比，每次比较后都把“较大者”存放在 max 中，全部元素比较完成后，max 的值就是所有元素中的最大值。

参考程序如下：

```
#include <stdio.h>
int max_value(int a[ ][4])
{
    int i, j,max;
    max=a[0][0];
    for (i=0;i<3;i++)
        for (j=0;j<4;j++)
            if (a[i][j]>max)
                max=a[i][j];
    return max;
}
int main( )
{
    int a[3][4]={{1,3,5,7},{2,4,6,8},{15,17,34,12}};
    int i,j;
    printf("The array is:\n");
    for (i=0;i<3;i++)                          // 输出矩阵
    {
        for (j=0;j<4;j++)
```

```
                printf("%–4d",a[i][j]);
            printf("\n");
        }
        printf("max value is %d\n",max_value(a));      // 输出最大值
        return 0;
}
```

4.5　字符串与指针

4.5.1　用字符指针访问字符串

C 语言中，用字符数组来存储字符串。在前面的学习中，可以利用循环对字符串中的每个字符依次进行存取，例如：

```
#include <stdio.h>
int main( )
{
    char a[ ]="Hello, World!", b[20];
    int i;
    for (i=0; a[i]!='\0'; i++)              // 把字符串 a 中的字符逐个赋值到数组 b 中
        b[i]=a[i];
    b[i]='\0';                              // 在 b 的末尾添加结束标志'\0'
    printf("%s\n", b);
    return 0;
}
```

从 4.4 节中指针与数组的关系可以看到，利用指针处理连续的内存单元是非常方便的。因此，也可以利用指针来进行字符串的处理。

在对字符串中的字符进行存取时，可以定义一个指针变量，指向字符串中的第 1 个字符（即赋值为字符数组首地址），再利用指针的移动来访问字符串中的字符。例如：

```
#include <stdio.h>
int main( )
{
    char a[ ]="Hello, World!",b[20];
    char *pa=a,*pb=b;
    for ( ;*pa!='\0'; pa++,pb++)            // 利用指针的移动依次访问各字符
        *pb=*pa;
    *pb='\0';
    printf("%s\n",b);
    return 0;
}
```

字符指针可以指向一个存储于数组中的字符串，也可以指向一个字符串常量。例如：

```
#include <stdio.h>
int main( )
{
    char *s="I love China!";
    printf("%s\n",s);
    return 0;
}
```

在编译时，系统为字符串常量分配一段内存空间。上例中指针 s 就指向这一段内存空间的第 1 个内存单元，即字符“I”，此时可以通过指针 s 访问字符串。

【例 4.14】 比较两个单词的大小（单词全部用小写字母）。按照字典排列法，前面的单词小，后面的单词大。

分析：定义两个指针变量，分别指向这两个单词，然后利用循环逐个字符进行比较，即从各自的第 1 个字母开始依次比较。如果两个字母相同，那么指针变量分别增 1，指向下一个字母，继续比较下一个字母；如果不同或字符串结束就可以结束循环，循环结束后比较使循环结束的那两个字母就可以区分出两个单词谁大谁小；如果循环结束后字符仍相同，则两个单词相等。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    char a[30],b[30],*sa,*sb;
    int i=0;
    sa=a; sb=b;                                  // sa 和 sb 分别指向字符串 a 和 b
    gets(sa);
    gets(sb);
    while ((*sa==*sb)&&*sa!='\0'&&*sb!='\0')     // 对应的字符相等且字符串均未结束
    {    sa++,sb++; }                            // 指针均增 1，继续指向下一个字符
    if (*sa=='\0' && *sb=='\0')                  // 两个单词同时结束，说明相等
        printf("%s and %s is equal!\n",sa,sb);
    else if (*sa<*sb)                            // sa 指向的字符小于 sb 指向的字符
        printf("max is %s,min is %s\n",sb,sa);
    else                                         // sa 指向的字符大于 sb 指向的字符
        printf("max is %s,min is %s\n",sa,sb);
    return 0;
}
```

4.5.2　字符指针和字符数组的区别

虽然字符指针和字符数组都可以用于字符串的处理，但两者是不同的。

字符数组在定义时，不论是否进行初始化，都会为其分配相应大小存储空间，以存储数组的内容，它存储的是字符串中的字符。

字符指针则不同，系统只分配一个指针变量的存储单元，用于存储其值（一个地址），如果没有初始化，其值不确定，指向不确定的内存单元；如果用一个字符串对其进行初始化，系统首先给字符串分配一段连续的内存单元，用于存放该字符串，然后将这段内存单元的首地址赋值给该指针变量。

字符数组一旦定义，其使用的存储空间就是固定的，在任何时候都可以使用数组名对数组进行访问。

字符指针变量只是一个指向内存单元的指针，在程序中改变后将不再指向原来的字符串。

例如：

```
#include <stdio.h>
int main( )
{
    char str[ ]="this is a test!";          // 字符数组
    char *p;                                // 字符指针
    int i=0;
    p=str;
    while (*p!='\0')
    {
        printf("%c",*p);
        p++;
        printf("%c",str[i]);
        i++;
    }
    printf("%s\n",p);                       // 由于指针 p 已被改变，所以不会输出原字符串
    printf("%s\n",str);                     // 使用 str 来输出字符串
```

```
    return 0;
}
```

字符数组只能在初始化时进行字符串的整体赋值，在程序运行中则不能。例如：

```
char str[ ]="this is a test!";          // 初始化字符数组
str="Hello world!";                     // 错误的用法
```

字符指针变量既可以在初始化时用字符串赋值，又可以在程序运行中进行赋值。在程序中，赋值后指针变量将指向所赋的字符串，而不再指向原字符串。例如：

```
char *str="this is a test!";    // 初始化时赋值，指向字符串"this is a test!"在内存的起始地址
str="Hello world!";             // 此时将指向字符串"Hello world!"在内存的起始地址
```

字符数组名是一个常量，在程序中只能引用，不能改变。字符指针是一个变量，在程序中可以指向任一位置，因此应该注意指针的位置，以防止引用出错。

字符数组与字符指针的区别如下：

① 字符数组由若干个元素组成，每个元素中存放 1 个字符，而字符指针变量中存放的是地址，不是将字符串放到字符指针变量中。

② 对于字符数组只能对各个元素赋值。例如：

```
char str[20];
str[20]="I love China!";                // 错误，不能直接赋值
```

而对于字符指针可进行整体赋值，例如：

```
char *ps;
ps="I love China!";
```

③ 对字符指针变量赋初值，例如：

```
char *s="I love China!";
```

等价于：

```
char *s;
s="I love China!";
```

对字符数组赋初值，例如：

```
char str[ ]="I love China!";
```

等价于：

```
char str[14];
strcpy(str,"I love China!");            // 只能使用字符串复制函数
```

④ 如果定义了一个字符数组，在编译时系统为它分配了内存单元，它有确定的地址，此时可以对数组名进行输入、赋值等操作。而定义一个字符指针变量时，系统给指针变量分配了内存单元，但指针没有指向一个确定的内存单元，此时不能对指针变量进行输入、赋值等操作。例如：

```
char str[10];
scanf("%s",str);                        // 正确，str 指向一段内存单元
```

以下与输入是不对的：

```
char *a;
scanf("%s",a);                          // 错误，a 没有指向确定的内存单元
```

原因是，scanf 的含义是将输入的字符串放入 a 所指向的内存单元中，但此时指针变量 a 并没有指向某个确定的内存单元，所以不能完成输入。可改为：

```
char *a,str[10];
a=str;
scanf("%s",a);
```

即先使 a 有一个确定值，指向一段确定的内存单元，如使 a 指向一个数组，然后输入一个字符串，把它存放在以该地址开始的内存单元中。

⑤ 指针变量的值是可以改变的。例如：

```
#include <stdio.h>
int main( )
```

```
{
    char str[ ]="I love China!",*p=str;
    printf("%s\n",p);
    p=p+7;
    printf("%s\n",p);
    return 0;
}
```

4.5.3　字符串作为函数参数

将一个字符串从一个函数传递到另一个函数，可以用地址传递的方法，即用字符数组名或指向字符串的指针变量作为参数。在被调用的函数中，可以改变字符串的内容，在主调函数中可以得到改变了的字符串。

【例 4.15】 用函数调用实现字符串的复制。

分析：要实现字符串的复制，可对字符串中的每个字符进行一个一个的复制，用字符串结束标志'\0'来控制循环。要注意，产生的新串也必须要有串结束标志'\0'，如果没有，要手动加上。

可以将复制的操作用函数实现，有以下几种形式。

（1）用字符数组作为参数

程序如下：

```
#include <stdio.h>
void copy_string(char to[ ], char from[ ])
{
    int i=0;
    while (from[i]!='\0')          // 判断字符串是否结束
    {
        to[i]=from[i];             // 复制对应的字符
        i++;
    }
    to[i]='\0';                    // 在末尾加上结束标志'\0'
}
int main( )
{
    char a[ ]="abcdefg";
    char b[ ]="12345";
    …
    copy_string(a,b);
    …
    return 0;
}
```

在 main 函数中也可以不定义字符数组，而用字符型指针变量，程序如下：

```
int main( )
{
    char a[ ]="abcdefg";
    char *b="12345";
    …
    copy_string(a,b);
    …
    return 0;
}
```

（2）形参用字符指针变量

程序如下：

```
#include <stdio.h>
void copy_string(char *to, char *from)
```

```
{
    for (;*from!='\0';from++,to++)
        *to=*from;
    *to='\0';                          // 注意：不能漏了'\0'
}
int main( )
{
    char a[ ]="abcdefg";
    char *b="12345";
    …
    copy_string(a,b);
    …
    return 0;
}
```

可以对 copy_string 函数做下列简化：

① 形式一：

```
void copy_string(char *to, char *from)
{
    while ((*to=*from)!='\0')          // 等价于：while (*to=*from)
    {
        to++;
        from++;
    }
}
```

思考：为什么循环语句后没有赋值语句“*to='\0';”？

② 形式二：

```
void copy_string(char *to, char *from)
{
    while ((*to++=*from++)!='\0');     // 等价于：while (*to++=*from++)
}
```

③ 形式三：

```
void copy_string(char *to, char *from)
{
    while (*from!='\0' )               // 等价于：while (*from)
        *to++=*from++;
    *to='\0';
}
```

④ 形式四：

```
void copy_string(char *to, char *from)
{
    for (;(*to++=*from++)!='\0';)
        ;
}
```

⑤ 形式五：

```
void copy_string(char *to, char *from)
{
    for (;(*to++=*from++);)
        ;
}
```

4.6　典 型 例 题

【例 4.16】 冒泡法排序：输入一个班级 20 名学生的 C 语言考试成绩，按成绩从高到低的顺序排序。

分析：冒泡法排序的思路是：将相邻的两个数比较，将大数换到前面。

本题中有 20 个成绩，第 1 趟排序时，依次将第 1 个数与第 2 个数比较、第 2 个数与第 3 个数比较、……、第 19 个数与 20 个数比较，共比较 19 次。每次比较时，如果前面的数小于后面的数，则进行交换，把大数换到前面，小数换到后面。经过第一趟比较后，可以找到最小数，换到最后。第 2 趟排序时，依次将第 1 个数与第 2 个数比较、……、第 18 个数与 19 个数比较，共比较 18 次，找到次小数。依次类推，第 19 趟排序只需比较 1 次。排完 20 个数，一共需经过 19 趟排序。由此可知，如果有 n 个数，共需经过 n−1 趟排序，在第 1 趟排序中要进行 n−1 次两两比较，在第 i 趟排序中要进行 n−i 次两两比较。

参考程序如下：

```
#include <stdio.h>
#define N 20
int main( )
{
    int x[N],temp;
    int i,j;
    printf("Input 20 scores:\n");
    for (i=0;i<N;i++)                    // 输入 20 个成绩
        scanf("%d",&x[i]);
    for (i=0;i<N-1;i++)                  // 冒泡法排序，i 表示排序趟数
        for (j=0;j<N-1-i;j++)            // j 表示每趟排序比较的次数
            if (x[j]<x[j+1])             // 前面元素小则交换到后面
            {
                temp=x[j];
                x[j]=x[j+1];
                x[j+1]=temp;
            }
    printf("The sorted score is:\n");
    for (i=0;i<N;i++)                    // 输出排序后的成绩
        printf("%d ",x[i]);
    return 0;
}
```

【例 4.17】 选择法排序：对 10 个整数按从小到大的顺序排序。

分析：选择法排序的思路是：第 1 趟排序时，找出 10 个数中的最小数，与第 1 个数交换位置。第 2 趟排序时，找出剩下 9 个数中的最小数，与第 2 个数交换位置。……，依次类推，第 9 趟排序时，找出剩下 2 个数中的最小数，与第 9 个数交换位置。如果有 n 个数，经过 n−1 趟排序后，即可得到从小到大的序列。

参考程序如下：

```
#include <stdio.h>
#define N 10
int main( )
{
    int i,j,min,temp,a[N+1];
    printf("Input %d data:\n",N);
    for (i=1;i<=N;i++)                   // 输入 N 个数
        scanf("%d",&a[i]);
    printf("The source numbers:\n");
    for (i=1;i<=N;i++)                   // 输出排序之前的 N 个数
        printf("%d ",a[i]);
    printf("\n");
```

```
    for (i=1;i<=N-1;i++)                    // 选择法排序，需要排 n-1 趟
    {
        min=i;                              // min 用于存放最小数的下标
        for (j=i+1;j<=N;j++)                // 找最小数，并记录其下标
            if (a[min]>a[j])
                min=j;
        temp=a[i];                          // 将 a[i]与最小数 a[min]交换
        a[i]=a[min];
        a[min]=temp;
    }
    printf("The sorted numbers:\n");
    for (i=1;i<=N;i++)                      // 输出已排好序的 N 个数
        printf("%d ",a[i]);
    printf("\n");
    return 0;
}
```

【例 4.18】 折半查找：有 15 个数按从小到大的顺序存放在一个数组中，输入一个数，要求找出该数在数组中的序号。如果该数不在数组中，则输出无此数。

分析：从表列中查找一个数最简单的方法是从第 1 个数开始顺序查找，将要找的数与表列中的数一一比较，直到找到为止。这种顺序查找法效率较低。

折半查找法是效率较高的一种方法。基本思路是，将输入的数与序列中居中的数进行比较，若小于中间数，则表示输入的数应该在前半序列中，再将该数与前半序列的中间数进行比较，根据大小关系确定输入的数是处于前半序列还是后半序列。这样重复下去，通过不断缩小范围，直到查找结束。

参考程序如下：

```
#include <stdio.h>
#define N 15
int main( )
{
    int i,j,number,top,bott,mid,loc,a[N],flag,sign=1;
    char c;
    printf("Enter 1 data:");                    // 输入第 1 个数
    scanf("%d",&a[0]);
    i=1;
    while (i<N)
    {
        printf("Enter %d data:",i+1);           // 继续输入后面的数
        scanf("%d",&a[i]);
        if (a[i]>=a[i-1])                       // 检查是否大于等于上一个数，否则重新输入
            i++;
        else
            printf("Enter a larger data:");
    }
    printf("The sort is:\n");
    for (i=0;i<N;i++)
        printf("%-4d",a[i]);
    printf("\n");
    flag=1;
    while (flag)
    {
        printf("Input number to look for:");
        scanf("%d",&number);                    // 输入要查找的数
```

```
            loc=0;
            bott=0;                                  // 确定查找范围的下限
            top=N–1;                                 // 确定查找范围的上限
            if ((number<a[0])||(number>a[N–1]))
                loc=–1;
            while ((sign==1)&&(bott<=top))
            {
                mid=(bott+top)/2;                    // 计算待查范围中间元素的下标
                if (number==a[mid])                  // 相等则表示找到该数
                {
                    loc=mid;
                    printf("Find %d,its position is %d\n",number,loc+1);
                    sign=0;
                }
                else if (number<a[mid])              // 小于中间数则查找范围上限变为 mid–1
                    top=mid–1;
                else                                 // 大于中间数则查找范围下限变为 mid+1
                    bott=mid+1;
            }
            if (sign==1||loc==–1)
                printf("%d is not found.\n",number);
            printf("Continue or not(Y/N)?");         // 是否继续查找下一个数
            getchar( );
            scanf("%c",&c);
            if(c=='N'||c=='n')
                flag=0;
            else
                sign=1;
        }
        return 0;
    }
```

【例 4.19】 贪心法：设计一个找零钱的算法。要求从面值最大的币种开始，只有不能再找到，才找下一个面值的币种。

分析：该算法需要从问题的初始解出发，一步一步地接近给定目标，并尽可能快地去逼近更好的解，这种方法称为“贪心法”。贪心法追求最快、最优解，不能得到全部解。

参考程序如下：

```
#include<stdio.h>
#include <conio.h>
#define N 20                                 // 限制找零的最大张数
int main( )
{
    int find(int n,int *d,int c,int *pd);
    int n,k,i, p[N],d[7]={100,50,20,10,5,2,1};
    printf("Plase input moneny n:");
    scanf("%d",&n);
    for (i=0;i<7;i++)                        // 判断找零的首张币种面额
        if (n>=d[i])
            break;
    k=find(n,&d[i],7–i,p);                   // 7–i 代表找零的币种数
    if (k<=0)
        printf("Error,Sorry!\n");
    else                                     // 输出 p[k]数组
    {
        printf("%d=%d",n,p[0]);              // 处理只找一张币的情况
        for (i=1;i<k;i++)
            printf("+%d",p[i]);
```

```
        }
        printf("\n");
        return 0;
}
int find(int n,int *d,int c,int *pd)          // 产生 p 数组，并统计找零币的张数
{
        int r;
        if (n==0) return 0;
        if (c==0) return -1;
        if (n<*d)
                return find(n,d+1,c-1,pd);    // 找下一个新的币种
        else                                  // 找同面额币种，放入 pd 中
        {
                *pd=*d;
                r=find(n-*d,d,c,pd+1);
                if (r>=0)
                        return r+1;
                return -1;
        }
}
```

【例 4.20】 某歌手大赛，共有 10 个评委给歌手打分，分数采用百分制，去掉一个最高分，去掉一个最低分，然后取平均分，得到歌手的成绩。10 个分数由键盘输入，编写程序计算某歌手的成绩。

参考程序如下：

```
#include <stdio.h>
float calculates(float s[ ],int n)
{
        int i;
        float max=s[0],min=s[0],sum=0,ave;
        for (i=0;i<n;i++)
        {
                if (s[i]>max)
                        max=s[i];
                if (s[i]<min)
                        min=s[i];
                sum=sum+s[i];
        }
        ave=(sum-max-min)/(n-2);
        return(ave);
}
int main( )
{
        int i;
        float score,s[10];
        printf("Please input the 10 scores:\n");
        for (i=0;i<10;i++)
                scanf("%f",&s[i]);
        score=calculates(s);
        printf("The singer's score is %.2f\n",score);
        return 0;
}
```

【例 4.21】 编程输出如下的斜塔形数字。

```
1   3   6   10 15
2   5   9   14
4   8   13
7   12
11
```

分析：该三角形为矩阵的上斜三角，除第 1 列外，其余元素都有规律可循。设行数为 i，列数为 j，矩阵数值存储于数组 t 中，矩阵有 k 层。同一行中，第 j 列的值等于同行中 j–1 列的值与列数及行数之和加 1，即 t[i][j]=t[i][j–1]+j+i+1，因此求出第 1 列就能计算出所有的值。第 1 列中，第 i 行的值等于同列中 i–1 行的值和行数之和，即 t[i][0]=t[i–1][0]+1。由于矩阵为上斜三角，当元素的行列之和等于矩阵层数时就应该换行，但由于 C 语言中的数组下界从 0 开始，因此层数应该减 1，即 i+j==k–1 时换行。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int t[40][40];
    int k,j,i;
    t[0][0]=1;                          // 给左上角元素赋值
    printf("Please input the layer(<20):");
    scanf("%d",&k);
    for (i=0;i<k;i++)
    {
        if (i>0)                        // 对第 1 列除左上角以外的元素赋值
            t[i][0]=t[i–1][0]+i;
        for (j=0;j<k;j++)
        {
            if (j>0)                    // 对第 i 行除第 1 列外的元素赋值
                t[i][j]=t[i][j–1]+j+i+1;
            if (j+i==k)                 // 当超出上斜三角范围时停止赋值
                break;
        }
    }
    for (i=0;i<k;i++)
    {
        for (j=0;j<k;j++)
        {
            if (j+i==k)                 // 当超出上斜三角范围时换到下一行
                break;
            printf("%–3d",t[i][j]);
        }
        printf("\n");
    }
    return 0;
}
```

【例 4.22】 有一行电文，已按如下规律译成密码：

A→Z　a→z
B→Y　b→y
C→X　c→x
…

即第 1 个字母变成第 26 个字母，第 *i* 个字母变成第（26–*i*+1）个字母。非字母字符不变。编写程序将密文译成原文。

分析：将字符存放到数组 s 中，如果 s[j]是大写字母，则它是第（s[j] –64）个大写字母。例如，s[j]的值是大写字母'B'，它的 ASCII 码为 66，它应是第（66–64）=2 个大写字母。按密码规定，应将它转换为第（26–i+1）个大写字母，即第（26–2+1）=25 个大写字母。而 26–i+1= 26–(s[j] –64)+1=26+64–s[j]+1，即 91–s[j]（如 s[j]等于'B'，91–'B'=91–66=25，s[j]应与第 25 个大写字母对换）。该字母的 ASCII 码为 91–s[j]+64，即 25+64=89，89 是'Y'的 ASCII 码。可以表示为 155–s[j]。小写字母的情况与此相似，但由于'a'的 ASCII 码为 97，因此，公式应改为 26+96–s[j]+1+96= 123–s[j]+96=219–s[j]。若 s[j]

的值为'b'，则其交换对象为 219–'b'=219–98=121，它是'y'的 ASCII 码。

由于此密码的规律是对称交换，即第 1 个字母与最后 1 个字母交换，第 2 个字母与倒数第 2 个字母交换，……。因此，从原文译为密文和从密文译为原文都是利用同一个公式。

参考程序如下：

```
#include <stdio.h>
int main( )
{
    int j,n;
    char s[80],tran[80];
    printf("Input cipher code:");
    gets(s);
    printf("cipher code:%s\n",s);
    j=0;
    while (s[j]!='\0')
    {
        if ((s[j]>='A')&&(s[j]<='Z'))
            tran[j]=155–s[j];
        else if ((s[j]>='a')&&(s[j]<='z'))
            tran[j]=219–s[j];
        else
            tran[j]=s[j];
        j++;
    }
    n=j;
    printf("original text:");
    for (j=0;j<n;j++)
        putchar(tran[j]);
    printf("\n");
    return 0;
}
```

运行结果如下：

```
Input cipher code:R droo erhrg Xsrmz mvcg dvvp. ↙
cipher code:R droo erhrg Xsrmz mvcg dvvp.
Original text:I will visit China next week.
```

【例 4.23】 有 *n* 个人围成一圈，顺序排号。从第一个人开始报数（从 1 报到 3），凡报到 3 的人退出圈子，问最后留下的是原来的第几号。

分析：本题使用数组非常方便，定义一个数组，将 1～*n* 放入数组，利用指针的移动，选择对应元素。退出的人将其对应的元素置 0，每次计数时排除为 0 的元素，最后不为 0 的元素就是剩下的人。

参考程序如下：

```
#include <stdio.h>
#define N 50
int main( )
{
    int i,k,m,n,num[N],*p;
    printf("Input number of person: n=");
    scanf("%d",&n);
    p=num;
    for (i=0;i<n;i++)
        *(p+i)=i+1;          // 以 1～n 为序给每个人编号
    i=0;                     // i 为每次循环时的计数变量
    k=0;                     // k 为按 1、2、3 报数时的计数变量
    m=0;                     // m 为退出人数
    while (m<n–1)   // 当退出人数比 n–1 少时（即未退出人数大于 1 时）执行循环
    {
```

```
            if (*(p+i)!=0)
                k++;
            if (k==3)          // 对退出的人的编号置为 0
            {
                *(p+i)=0;      // 对退出的人的编号置为 0
                k=0;           // 重新从 0 开始计数
                m++;           // 退出人数加 1
            }
            i++;
            if (i==n) i=0;     // 报数到尾后，i 恢复为 0
        }
        while (*p==0) p++;     // 查找不为 0 的元素，即还留在圈中的人
        printf("The last one is NO.%d\n",*p);
        return 0;
    }
```

习　　题

一、选择题

1．在 C 语言中，引用数组元素时，其数组下标的数据类型不允许是（　　）。

A）整型常量　　B）整型表达式

C）整型常量或整型表达式　　D）任何类型的表达式

2．以下对一维整型数组 a 的正确说明是（　　）。

A）int a(10);　　B）int n=10,a[n];

C）int n;　　D）#define SIZE 10

scanf("%d",&n);　　int a[SIZE];

int a[n];

3．若有说明“int a[10];”，则对 a 数组元素的正确引用是（　　）。

A）a[6−5]　　B）a(8)　　C）a[3.5]　　D）a[10]

4．若要定义一个具有 5 个元素的整型数组，以下定义语句错误的是（　　）。

A）int a[5]={0};　　B）int b[ ]={0,0,0,0,0};

C）int c[2+3];　　D）int i=5,d[i];

5．若有定义“int a[5];”，则 a 数组中首元素的地址可以表示为（　　）。

A）&a　　B）a+1　　C）a　　D）&a[l]

6．若要对一维数组 a 的所有元素的值都初始化为 1 的语句正确的是（　　）。

A）int a[5]=(l,1,1,l,l);　　B）int a[ ]={l,l,l,l,l};

C）int a[ ]={5*1};　　D）int a[5]={1};

7．若有说明语句“int a[2][4];”，则对 a 数组元素引用正确的是（　　）。

A）a[0][3]　　B）a[0][4]　　C）a[2][2]　　D）a[2][2+1]

8．若有说明语句“int y[ ][4]={0,0};”，则下面叙述不正确的是（　　）。

A）数组 y 的每个元素都可得到初值 0

B）二维数组 y 的行数为 1

C）该说明等价于“int y[ ][4]={0};”

D）只有元素 y[0][0]和 y[0][1]可得到初值 0，其余元素均得不到初值 0

9．若有说明语句“int a[ ][3]={1,2,3,4,5,6,7,8};”，则 a 数组的行数为（　　）。

A）3　　B）2　　C）1　　D）不能确定

10．有以下的程序段：

```
char a[3],b[ ]="China";
a=b;
printf("%s\n",a）;
```

则（　　）。

A）运行后将输出 China　　B）运行后将输出 Ch

C）运行后将输出 Chi　　D）编译出错

11．下列选项中，能够满足“若字符串 s1 等于字符串 s2，则执行 ST”要求的是（　　）。

A）if (strcmp(s2,s1)==0) ST;　　B）if (sl==s2) ST;

C）if (strcmp(sl,s2)==1) ST;　　D）if (sl−s2==0) ST;

12．若用数组名作为函数调用时的实参，则实际上传递给形参的是（　　）。

A）数组首地址　　B）数组第一个元素的值

C）数组中全部元素的值　　D）数组中元素的个数

13．若使用一维数组名作为函数实参，则以下说法正确的是（　　）。

A）必须在被调函数中说明形参数组的大小

B）形参数组类型与实参数组类型可以不同

C）在被调函数中，形参数组的大小可以小于实参数组

D）实参数组名与形参数组名必须一致

14．若有以下定义，则∗(p+5)表示（　　）。

```
int a[10],∗p=a;
```

A）元素 a[5]的地址　　B）元素 a[5]的值

C）元素 a[6]的地址　　D）元素 a[6]的值

15．有如下说明：

```
int a[10]={1,2,3,4,5,6,7,8,9,10}, *p=a+3;
```

则数值为 9 的表达式是（　　）。

A）*(p+5)　　B）*p+9　　C）*(p+8)　　D）*p+=9

16．设有说明“char s[20]="China"; *p=s;”，则以下不正确的表达式是（　　）。

A）p=s+1　　B）s=p+2　　C）s[2]=p[4]　　D）p=s[10]

17．下列语句组中，正确的是（　　）。

A）char ∗s; s="Olympic";　　B）char s[7]; s="Olympic";

C）char ∗s; *s="Olympic";　　D）char s[7]; *s="Olympic";

18．设有以下程序段：

```
char s[ ]="China";
char ∗p;
p=s;
```

则下列叙述中正确的是（　　）。

A）s 和 p 完全相同

B）数组 s 中的内容和指针变量 p 中的内容相同

C）∗p 与 s[0]相等

D）s 数组长度和 p 所指向的字符串长度相等

19．以下函数 findmax 拟实现在数组中查找最大值并作为函数值返回，但程序中有错导致不能实

现预定功能。

```
#define MIN −2147463647
int findmax(int x[ ],int n)
{
    int i,max;
    for (i=0;i<n;i++)
    {
        max=MIN;
        if (max<x[i])
            max=x[i];
    }
    return max;
}
```

造成错误的原因是（　　）。

A）定义语句“int i,max;”中 max 未赋值

B）赋值语句“max=MIN;”中，不应该给 max 赋 MIN 值

C）语句“if (max<x[i])　max=x[i];”中判断条件设置错误

D）赋值语句“max=MIN;”放错了位置

20．有以下程序段：

```
char s[10]="abcd";
printf("%d,%d\n",strlen(s),sizeof(s));
```

执行后的输出结果是（　　）。

A）4,5　　B）4,10　　C）5,10　　D）10,10

21．有以下程序段：

```
char s[ ]="ABCD",*p;
for (p=s;p<s+4;p++)
    printf("%s",p);
```

执行后的输出结果是（　　）。

A）ABCDBCDCDD　　B）ABCD

C）DCBA　　D）ABCDABCABA

22．以下程序段的运行结果是（　　）。

```
char a[ ]="program",*p;
p=a;
while (*p!='g')
{
    printf("%c",*p−32);
    p++;
}
```

A）PROgram　　B）PROGRAM　　C）PRO　　D）proGRAM

二、填空题

1．数组是一组具有相同_____、固定_____的元素组成的数据集合。

2．在 C 语言中，数组下标的取值是从_____开始的。

3．若有定义“int a[10];”，则数组名 a 实际上是代表该数组的_____。

4．在 C 语言中，二维数组在内存中占用一段连续的内存单元，数组元素的存放顺序是_____。

5．若有定义“int a[3][6];”，按在内存中的存放顺序，a 数组的第 10 个元素是_____。

6．若有定义“double x[3][5];”，则 x 数组中行下标的最小值为_____，列下标的最大值为_____。

7．若有定义“int a[3][4]={{1,2},{0},{4,6,8,10}};”，则 a[1][2]的值是_____，a[2][1]的值是_____。

8．如果在为数组赋初值时，没有足够的数据赋给数组元素，则对于整型/实型数组，未赋值的数组元素值为_____，对于字符数组则为_____。

9．若有定义“char str[10];”，用 str 存储一个字符串时，则该字符串的最大长度为_____。

10．若要在程序中使用 strcpy 函数，则应该在程序开头用#include 命令包含的头文件是_____。

11．若有定义“char *s="ABCD";”，则执行语句“printf("%s\n",s);”的输出结果是_____，执行语句“printf("%c\n",*s);”的输出结果是_____。

12．'A'在内存中存放时将占用_____个字节，"A"在内存中存放时将占用_____个字节。

13．数组名作为函数实参时，被调函数的形参可以定义成_____或_____的形式。

14．若有定义“int a[10],*p=&a[3];”，则 p[4]表示的数组元素是_____。

15．若有定义“double x[10],*p1=&x[5],*p2=x;”，则表达式“p1−p2”的值为_____。

三、程序填空题

1．下面程序的运行结果是“3 5 8”，请填空。

```
#include <stdio.h>
int main( )
{
    int x[5],i;
    x[0]=l; x[l]=2;
    for (i=2;i<5;i++)
        x[i]=_____;
    for (i=_____;i<5;i++)
     printf("%d ",x[i]);
    return 0;
}
```

2．以下程序是用冒泡法对数组的各元素进行降序排序，请填空。

```
#include <stdio.h>
int main( )
{
    int a[10],i,j,t;
    for (i=0;i<10;i++)
        scanf("%d",_____);
    for (i=0;i<9;i++)
        for (j=0; _____;j++)
            if (_____)
            {
                t=a[j];
                a[j]=a[j+1];
                a[j+1]=t;
            }
    for (i=0;i<10;i++)
        printf("a[%d]=%d\n",i,a[i]);
    return 0;
}
```

3．有以下程序：

```
#include <stdio.h>
int main( )
{
    int a[3][3]={{1,2,3},{4,5,6},{7,8,9}};
```

```
    int b[3]={0},i;
    for (i=0;i<3;i++)
        b[i]=a[i][2]+a[2][i];
    for (i=0;i<3;i++)
        printf("%d",b[i]);
    printf("\n");
    return 0;
}
```

程序运行后的输出结果是_____。

4．以下程序是求矩阵 a、b 的和，结果存入矩阵 c 中并按矩阵形式输出，请填空。

```
#include <stdio.h>
int main( )
{
    int a[3][4]={{3,−2,7,5},{1,0,4,−3},{3,0,5,1}};
    int b[3][4]={{−2,0,1,4},{5,−1,7,6},{6,8,0,2}};
    int i,j,c[3][4];
    for (i=0;i<3;i++)
        for (j=0;j<4;j++)
            c[i][j]=_____;
    for (i=0;i<3;i++)
    {
        for (j=0;j<4;j++)
            printf("%3d",c[i][j]);
        _____;
    }
    return 0;
}
```

5．下列程序的运行结果是_____。

```
#include <stdio.h>
int main()
{
    char s[ ]="chinaren";
    int i=5;
    printf("%s\n",s+i);
    return 0;
}
```

6．以下程序段的运行结果是_____。

```
#include <stdio.h>
int main( )
{
    char s[ ]="600x1y2";
    int i,x=0;
    for (i=0;s[i]>='0'&&s[i]<='9';i++)
        x=10∗x+s[i]−'0';
    printf("%d\n",x);
    return 0;
}
```

7．有以下程序：

```
#include <stdio.h>
int main( )
{
    char s[30];
    scanf("%s",s);
    printf("%s\n",s);
    return 0;
}
```

程序运行时从键盘输入：How are you?↙

则输出结果为_____。

8．以下程序的功能是将两个字符串 s1 和 s2 连接起来，请填空。

```
#include <stdio.h>
void join(char *p1,char *p2)
{
    char *p=p1;
    while (_____)
        p++;
    for (;*p2!='\0',p++,p2++)
        *p=_____;
    *p='\0';
}
int main( )
{
    char s1[80],s2[30];
    gets(s1);
    gets(s2);
    join(s1,s2);
    puts(s1);
    return 0;
}
```

9．以下程序的运行结果是_____。

```
#include <stdio.h>
int fun(char *s)
{
    char *p=s;
    while (*p)
        p++;
    return(p-s);
}
int main( )
{
    char *t="abded";
    int x;
    x=fun(t);
    printf("%d\n",x);
    return 0;
}
```

10．下面程序的运行结果是_____。

```
#include <stdio.h>
#include <string.h>
void fun(char *w,int n)
{
    char t,*s1,*s2;
    s1=w;
    s2=w+n-1;
    while (s1<s2)
    {
        t=*s1; *s1=*s2; *s2=t;
        s1++;   s2--;
    }
}
int main( )
{
    char s[ ]="1234567";
```

```
    fun(s,strlen(s));
    puts(s);
    return 0;
}
```

四、编程题

1．从键盘输入 N 个数（在数组中从下标为 0 的位置开始存放），N 由下面的符号常量定义。编写函数 bubbleSort，用冒泡法将下标为 m 到下标为 n 的数按降序排序。这 N 个数的输入/输出及 m、n 的输入都在主函数中完成。设 $0 \leqslant m < N-1$，$m < n \leqslant N-1$。

```
#define N 15
```

2．设有一个已按从大到小顺序排好的数列存放在一维数组中，依次为 81,76,66,61,54,47,36,30,22,16。编写程序，输入一个数，仍按原来的排序规律将其插入到数组中。

3．输入一个 $N \times N$ 的矩阵，检查该矩阵是否为对称阵，输出判断结果。

4．找出一个二维数组中的鞍点，即该位置上的元素在该行上最大，在该列上最小。也可能没有鞍点。

5．编写函数，将一个无符号十进制整数转化为二进制形式，保存在形参数组中（在主函数中输出其二进制形式）。

6．编写函数，将一个整型数转换成对应的字符串。例如输入 824734，应输出字符串“824734”。要求使用字符串作为函数参数，输入输出均在主函数中完成。

第 5 章　复杂构造数据类型

通过第 4 章的学习，我们知道对于同种类型的数据可以用数组来存储和访问，数组的使用既保证了数据的整体性，又提高了数据处理的效率。但在解决实际问题的过程中，还会遇到这样一种问题：在描述同一个对象时，数据的类型不一致。例如，学生个人信息包括姓名、学号、年龄、各门课程的考试成绩等，显然是一组有内在关联的数据。这就需要采用本章将要介绍的结构体数据类型。

本章介绍复杂构造数据类型的编程；主要介绍结构体类型的定义与使用，简单介绍共用体和枚举类型应用；最后简要介绍链表。

5.1　结　构　体

表 5.1 中，一行表示一个学生的信息。表 5.2 中，一行表示一个职工的工资信息，这样的一行信息称为一条记录。计算机在进行信息处理时，通常都是以记录为单位进行处理的。

观察发现，在一条记录中既有整型数据（学号、编号、成绩等）、实型数据（工资等），又有字符串（姓名）。表示一个学生的信息或一个职工的工资信息，都包括若干种不同类型的数据，即一条记录往往由多种不同类型的数据组成。对于一组相互关联但不同类型的数据，按照以前的方法是无法组合在一起的，也无法处理这种复杂数据结构。

表 5.1　学生成绩表

学　号	姓　名	语　文	数　学	英　语	总　分
1001	周炎	78	62	67	207
1002	王林	87	73	86	246
1003	丁鹏	65	85	93	243
1004	李娟	68	82	81	231
……	……	……	……	……	……

表 5.2　职工工资表

编　号	姓　名	基本工资	奖　金	保　险	实发工资
1	刘俊	500	600	100	1000
2	王军	500	400	100	800
3	丁建	500	700	100	1100
4	李文	500	1100	200	1400
……	……	……	……	……	……

为了处理这类内部有关联的一组数据，需要定义新的数据类型。在 C 语言中这种新的数据类型称为结构体。

5.1.1　结构体的引出及使用

1．结构体的引出

【例 5.1】 职工工资管理系统中每位职工的信息包括：编号、姓名、基本工资、奖金、保险和实发工资。输入每个职工的前 5 项信息，计算每个职工的实发工资，并依次输出每个职工的信息。

分析：每位职工共有 6 项基本信息，根据基本工资、奖金、保险计算出第 6 项实发工资（实发工资=基本工资+奖金–保险）。由于各个数据类型不同，编号 num 为整型，姓名 name 为字符数组（即字符串），基本工资、奖金、实发工资为实型，可以使用结构体类型描述这些数据。

将表示职工的信息定义为一个结构体类型，设结构体类型名为 struct worker，包含 6 个成员，定义语句为：

```
struct worker
{
    int num;                    // 编号
    char name[20];              // 姓名
    float pay;                  // 基本工资
    float bonus;                // 奖金
    float insurance;            // 保险
    float realpay;              // 实发工资
};
```

上面的定义中，struct worker 是一个自定义的结构体类型，它与系统的标准类型（如 int、char、float 等）具有相同的作用。不过 int 和 char 是系统给出的，而 struct worker 是程序员根据需要自己定义的。

为了能存放一个职工的信息，还需要定义该结构体类型（struct worker）的变量，定义一个结构体变量 w 的语句为：

```
struct worker w;
```

使用结构体，实际上是通过结构体变量引用它的成员。对变量 w 来说，它的 num、name、pay 等 6 项就是 w 的成员。引用成员时，要在变量名与成员名中间加一个小数点，例如 w.num=1001。

参考程序如下：

```
#include <stdio.h>
struct worker                   // 定义结构体类型
{
    int num;
    char name[20];
    float pay;
    float bonus;
    float insurance;
    float realpay;
};
int main( )
{
    struct worker w;            // 定义结构体变量
    int i;
    for (i=0;i<10;i++)          // 逐个输入每个职工的各项信息
    {
        printf("num:");
        scanf("%d",&w.num);
        getchar( );             // 在使用 gets 函数之前将上次的回车读走
        printf("name:");
        gets(w.name);
        printf("pay:");
        scanf("%f",&w.pay);
        printf("bonus:");
        scanf("%f",&w.bonus);
        printf("insurance:");
        scanf("%f",&w.insurance);
        w.realpay=w.pay+w.bonus−w.insurance;          // 计算该职工的实发工资
        printf("num: %d\tname: %s\tpay: %f\tbonus: %f\tinsurance:%f\trealpay:%f\n",w.num,w.name,
            w.pay,w. bonus,w. insurance,w.realpay);   // 打印该名职工的全部信息
}
```

```
return 0;
}
```

2．结构体的定义

定义结构体类型的一般形式为：

```
struct 结构体名
{
        类型标识符 1 成员名 1;
        类型标识符 2 成员名 2;
        …
        类型标识符 n 成员名 n;
};
```

例如：

```
struct student                  // 定义学生结构体类型
{
    int num;                    // 学号
    char name[20];              // 姓名
    char sex;                   // 性别
    int age;                    // 年龄
    float score;                // 成绩
    char addr[30];              // 地址
};
```

结构体类型用 struct 关键字定义，花括号里的每一项称为结构体成员，结构体成员的类型可以是普通的数据类型（如 int、char 等），也可以是数组、指针或已经定义的结构体类型等。

结构体类型在使用之前应先定义其类型，再定义该类型的变量，然后才能使用。

3．定义结构体类型变量的方法

注意，前面定义的 struct worker 只是一种结构体类型，不能直接用来存放数据，需由继续定义此类型的结构体变量才能使用。

可以采用以下 3 种方法定义结构体类型变量。

（1）先定义结构体类型，再定义结构体变量

例如：

```
struct student                  // 定义结构体类型
{
  int num;
  char name[20];
  float score;
};
struct student stu1,stu2;       // 定义结构体变量
```

注意：关键字 struct 要与结构体名一起使用，共同构成结构体类型名。

定义了结构体变量后，系统分别为每个成员分配相应大小的内存单元，结构体的各成员在内存中是按定义的顺序连续存放的，所以结构体变量在内存中占据的字节数是各个成员的长度之和。

结构体类型变量在内存中所占据的字节数可用 sizeof 运算符求出：

```
sizeof(类型名)
```

例如，sizeof(float)为 4，sizeof(struct student)为 4+20+4=28。

（2）在定义结构体类型的同时定义变量

例如：

```
struct student
{
```

```
        int num;
        char name[20];
        float score;
    }stu1,stu2;
```

该定义形式中既定义了类型，又定义了变量。如果在后续程序中还需要用到该类型的变量，可以再进行定义，例如：

```
    struct student stu3;
```

（3）直接定义结构体类型变量

例如：

```
    struct
    {
        int num;
        char name[20];
        float score;
    }stu1, stu2;
```

该定义形式中省略了结构体名，则在后续程序中不能再定义该类型的变量，例如下面的定义是不合法的：

```
    struct stu3;
```

在结构体的定义中，成员的类型既可以是基本数据类型，也可以是结构体类型。当成员的类型是结构体类型时，就构成了结构体类型的嵌套定义。例如，学生的年龄可以改用出生日期（年、月、日）来表示，定义形式如下：

```
    struct date                          // 定义表示日期的结构体类型
    {
        int year;
        int month;
        int day;
    };
    struct student
    {
        int num;
        char name[20];
        char sex;
        struct date birth;               // 使用 struct date 定义表示生日的成员 birth
        char addr[30];
    };
```

则 sizeof(struct student)应为：4+20+1+(4+4+4)+30=67。

4．结构体类型变量的引用

使用结构体变量时，不能直接使用结构体变量来存取数据，而是引用结构体变量的成员来实现对结构体变量的使用。

（1）引用形式

结构体变量成员的引用形式如下：

```
    结构体变量名.成员名
```

例如，对于以下的结构体类型：

```
    struct student
    {
        int num;
        char name[20];
        float score;
    };
```

可以进行如下的定义和引用：

```
struct student stu1,stu2;
stu1.num=10001;
strcpy(stu1.name,"Li Ming");
stu1.score=95;
```

（2）结构体变量使用说明

① 结构体变量通常不能整体使用，如不能整体输入、输出，只能对单个成员分别引用。但当结构体变量作为函数参数或赋初值时，可以整体使用。或者两个相同类型的结构体变量，如果一个已经有值，可以对另外一个整体赋值。例如，上面的变量 stu1 已经赋值，则可以进行整体赋值，执行以下语句：

```
stu2=stu1;
```

等价于分别执行以下 3 个语句：

```
stu2.num=stu1.num;
strcpy(stu2.name,stu1.name);
stu2.score=stu1.score;
```

② 如果成员本身又属于一个结构体类型，则这个成员也不能整体输入输出，要用若干个成员运算符引用到最里层的成员，如前面的 birth 成员本身又是 struct date 类型的结构体变量，则成员引用形式如下：

```
scanf("%d",&stu1.birth.year);
scanf("%d",&stu1.birth.month);
printf("%d\n",stu1.birth.day);
```

③ 可以定义与结构体成员同名的普通变量，它们之间不会发生混乱。例如：

```
struct student stu;
int age,year;
…
stu.age=20;              // 引用结构体变量 stu 的成员 age
stu.birth.year=1980;
…
age=24;                  // 引用普通变量 age
year=2000;               // 引用普通变量 year
```

5．结构体变量的初始化

可以在定义结构体变量的同时，对其初始化。例如：

```
struct student
{
    int num;
    char name[20];
    float score;
}stu1={15001,"宋红",89.5};
```

还可以写成以下的形式：

```
struct student stu2={15001,"宋红",89.5};
```

注意，以下的赋值是错误的：

```
struct student stu3;
stu3={15001,"宋红",89.5};
```

5.1.2　结构体数组

结构体既然是一种数据类型，就可以定义以结构体为基类型的数组，数组中的每个元素都是结构体类型的变量。定义结构体数组的方法和定义结构体变量类似。例如：

```
struct student
{
    int num;
    char name[20];
    float score;
};
```

```
struct student stu[40];          // 定义结构体数组 stu，包含 40 个结构体变量
```

还可以定义为：

```
struct student
{
    int num;
    char name[20];
    float score;
}stu[40];
```

或者定义为：

```
struct
{
    int num;
    char name[20];
    float score;
}stu[40];
```

结构体数组的各元素在内存中是连续存放的，各元素的成员也按顺序存放，如图 5.1 所示。

	num	name	sex	score	addr
stu[0]	10101	Li Lin	M	87.5	103 Beijing
stu[1]	10102	Zu Feng	F	98	130 Shanghai
stu[2]	10103	Wang Mi	M	78	104 Jinan

图 5.1　学生具体信息举例

结构体数组名也是一个指针常量，表示该结构体数组在内存中存放的首地址。

与其他类型的数组一样，对结构体数组也可以进行初始化。例如：

```
struct student
{
    int num;
    char name[20];
    char sex;
    float score;
    char addr[20];
}stu[3]={{10101,"Li Lin",'M',87.5,"103 Beijing"},
         {10102,"Zhu Feng", 'F',98,"130 Shanghai"},
         {10103,"Wang Mi",'M',78,"104 Jinan"}};
```

赋初值时也可以不指定元素的个数，系统根据初值个数确定数组元素个数。例如：

```
struct student stu[ ]= {{10101,"Li Lin",'M',87.5,"103 Beijing"},
                        {10102,"Zhu Feng", 'F',98,"130 Shanghai"},
                        {10103,"Wang Mi",'M',78,"104 Jinan"}};
```

【例 5.2】 一个职工工资管理系统，其内容包括：编号、姓名、基本工资、奖金、保险。共有 10 位职工，求每位职工的实发工资，并打印所有职工的全部信息。

分析：职工的信息可以定义为结构体类型 struct worker，现在有 10 位职工，每位职工都是结构体 struct worker 类型，可以定义一个结构体数组，即数组中每一个元素可以存放一个职工的信息。

参考程序如下：

```
#include <stdio.h>
struct worker
{
    int num;
    char name[20];
    float pay;
    float bonus;
    float insurance;
```

```
    float realpay;
};
int main( )
{
    int i;
    struct worker w[10];
    for (i=0;i<10;i++)                          // 逐个输入每个工人的各项信息
    {
        printf("num:");
        scanf("%d",&w[i].num);
        getchar( );
        printf("name:");
        gets(w[i].name);
        printf("pay:");
        scanf("%f",&w[i].pay);
        printf("bonus:");
        scanf("%f",&w[i].bonus);
        printf("insurance:");
        scanf("%f",&w[i].insurance);
        w[i].realpay=w[i].pay+w[i].bonus-w[i].insurance;          // 计算实发工资
    }
    for (i=0;i<10;i++)                                            // 打印工人的信息
    {
        printf("num:%d\tname:%s\tpay:%f\tbonus:%f\tinsurance:%f\trealpay:%f\n",w[i].num,
            w[i].name,w[i].pay,w[i].bonus,w[i].insurance,w[i].realpay);
    }

}
```

5.1.3　结构体程序举例

【例 5.3】 定义一个结构体变量（包括年、月、日）。输入一个日期，计算该日在本年中是第几天。

分析：计算某天是一年中的第几天，需要考虑该天所在的月份，即用日期加上所在月份之前的各月的天数，再加上当前月的天数，即可得到所求。例如输入的是 3 月 15 日，则应该是本年的第（31+28+15=74）天。若是闰年，而且需要计算的日期是在 2 月份之后，则需要在算得的天数基础上再加 1 天。结构体变量 date 中的成员对应于输入的年、月、日。

参考程序如下：

```
#include <stdio.h>
struct
{
    int year;
    int month;
    int day;
}date;
int main( )
{
    int days;                       // 天数
    printf("Input year,month,day: ");
    scanf("%d.%d.%d",&date.year,&date.month,&date.day);
    switch(date.month)              // 根据月份计算前面各月包含的天数
    {
      case 1: days=date.day; break;
      case 2: days=date.day+31; break;
      case 3: days=date.day+59; break;
      case 4: days=date.day+90; break;
```

```
            case 5: days=date.day+120; break;
            case 6: days=date.day+151; break;
            case 7: days=date.day+181; break;
            case 8: days=date.day+212; break;
            case 9: days=date.day+243; break;
            case 10: days=date.day+273; break;
            case 11: days=date.day+304; break;
            case 12: days=date.day+334; break;
        }
        // 如果是闰年，且月份数大于等于 3，要多加 1 天
        if ((date.year%4==0&&date.year%100!=0||date.year%400==0)&&date.month>=3)
            days+=1;
        printf("%d/%d is the %dth day in %d.\n",date.month,date.day,days,date.year);
        return 0;
    }
```

运行结果如下：

```
Input year,month,day:2010.10.1↙
10/1 is the 274th day in 2010.
```

【例 5.4】 编写对候选人得票进行统计的程序。假设有 3 个候选人，每次输入一个得票的候选人的名字，最后输出各人的得票结果。假设共有 10 人投票。

分析：定义一个结构体数组，包含 3 个元素，表示有 3 个候选人。每个元素有两个成员，一个是名字，一个是得票数，得票数初始化为 0。根据输入的名字进行判断，输入谁的名字，就给相应元素的票数加 1。

参考程序如下：

```
#include <stdio.h>
#include <string.h>
struct person
{
    char name[20];
    int count;
};
int main( )
{
    struct person candidate[3]={{"Li",0},{"Zhang",0},{"Wang",0}}; // 定义结构数组并赋初值
    int i,j;
    char cname[20];                 // 定义字符数组，存放输入的候选人名字
    printf("Input the name in the ticket:\n");
    for (i=0;i<10;i++)              // 分别输入 10 张选票上的人名
    {
        scanf("%s",cname);      // 读入第 i 张选票上所选举的人名
        for (j=0;j<3;j++)
            if (strcmp(cname,candidate[j].name)==0)        // 选票上的名字与候选人名比较
                candidate[j].count++;
    }
    printf("\n");
    for (i=0;i<3;i++)
        printf("%5s: %d\n",candidate[i].name,candidate[i].count);
    return 0;
}
```

5.1.4　结构体与指针

结构体变量占有内存单元，具有相应的地址，一个结构体变量的地址称为这个结构体变量的指针。可以定义一个结构体类型的指针变量，存放结构体变量的地址，即指向相应的结构体变量。

1．指向结构体的指针

定义了指向结构体变量的指针后，就可以将结构体变量的地址赋给指针变量，则根据以前指针的定义，*p 就相当于它所指向的结构体变量。在进行成员引用时，可以用“(*p).”表示。这里要注意，( )不能省略，因为成员运算符“.”的优先级比指针运算符“*”更高。例如：

```
struct student
{
    int num;
    char name[20];
    float score;
}stu1,*p;
p=&stu1;
(*p).num=89101;                    // 等价于：stu1.num=89101;
strcpy((*p).name,"Li Lin");        // 等价于：strcpy(stu1.name,"Li Lin");
(*p).score=89;                     // 等价于：stu1.score=89;
```

注意：不能用指向结构体变量的指针指向该结构体变量的某个成员。例如，以下的赋值是错误的：

```
p=&stu1.num;
```

如果要访问 stu1.num，应定义一个 int 型指针变量指向该成员，例如：

```
int *ip;
ip=&stu1.num;
```

为了使用方便和直观，也可以用指向运算符“–>”，即用 p–>num 来代替(*p).num（指向运算符优先级也为 1）。因此，有 3 种引用结构体成员的方法：

① 结构体变量.成员名

② (*p).成员名

③ p–>成员名

思考：试分析以下几种运算符的结果和含义：

```
p–>n
p–>n++
++p–>n
(*p).num
```

于是，上面的程序也可以写为：

```
struct student
{
    int num;
    char name[20];
    float score;
}stu1,*p;
p=&stu1;
p–>num=89101;
strcpy(p–>name,"Li Lin");
p–>score=89;
```

结构体数组的每一个元素都是一个结构体变量，结构体变量在内存中是按顺序存放的。而数组元素在内存中也是按顺序存放的，即结构体数组在内存中存放时，先存放第 1 个数组元素（各成员按定义的顺序依次存放），再存放第 2 个数组元素（各成员按定义的顺序依次存放），……，依次类推。因此，只要知道结构体数组的首地址，即可依次找到结构体数组中每一个结构体变量的每个成员。根据指针的定义，也可以定义指向结构体数组的指针，即指向结构体数组的第一个元素。例如：

```
struct student
{
    int num;
```

```
    char name[20];
    float score;
};
struct student stu[3],*p;
…
for (p=stu;p<stu+3;p++)
    printf("%d,%s,%f\n",p–>num,p–>name,p–>score);
```

思考：程序的指针 p++后，p 增加的值是多少？

随着指针 p 指向的改变，以下结构体成员的引用所代表的信息也发生了变化（如图 5.2 所示）：

```
p–>num;    p–>name;    p–>score;
```

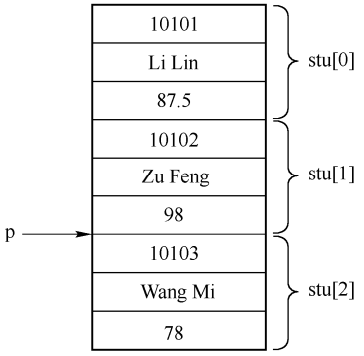

图 5.2　指向结构体数组的指针

注意：p 是指向结构体数组元素的，不能用来指向数组元素中的某一成员。例如“p=&stu[1].num”是错误的。

可以采用强制类型转换的方法，先将成员的地址转换成 p 的类型，如“p=(struct student *) stu[1].num;”，但通常不这样使用。

对指针变量进行赋值时，写成“p=&stu;”是不正确的，应写成“p=stu;”。因为 stu 本身就是一个地址，不能再加取地址符。

2．结构体作为函数参数

结构体变量也可以作为函数的参数。用结构体变量做参数时，由于形参也要占用内存单元（在函数被调用时分配，并在函数调用完毕后释放）。如果结构体类型复杂、规模大，函数调用在时间和空间上的开销都会很大。

因此，可以用指向结构体变量的指针做实参，将结构体变量（或结构体数组）的地址传递给被调函数中的形参，然后可以利用形参指针变量对结构体成员进行修改，并反映到主调函数。

【例 5.5】 有一个结构体变量 stu，内含学生学号、姓名和 3 门课的成绩，要求在 main 函数中赋值，在另一函数 print 中将它们打印输出。

根据题意，可定义如下结构体：

```
struct student
{
    int num;                // 学号
    char name[20];          // 姓名
    float score[3];         // 3 门课的成绩
};
```

定义一个结构体变量：“struct student stu;”，在进行参数传递时，可用该结构体变量作为参数，即把结构体变量的值传递给形参。

参考程序如下：

```
#include <stdio.h>
#include <string.h>
struct student
{
    int num;
    char name[20];
    float score[3];
};
void print(struct student stu)          // 形参定义为结构体变量
{
    printf("num:%d\nname:%s\nscore0:%.1f\nscore1:%.1f\nscore2:%.1f\n",
            stu.num,stu.name,stu.score[0],stu.score[1],stu.score[2]);
}
```

```
int main( )
{
    struct student stu;
    stu.num=12345;
    strcpy(stu.name,"Li Li");
    stu.score[0]=67.5;
    stu.score[1]=89;
    stu.score[2]=78;
    print(stu);                    // 用结构体变量作函数实参
    return 0;
}
```

改用指向结构体变量的指针作为实参，参考程序如下：

```
#include <stdio.h>
#include <string.h>
struct student
{
    int num;
    char name[20];
    float score[3];
};
void print(struct student *p)      // 形参定义为结构体指针
{
    printf("num:%d\nname:%s\nscore0:%.1f\nscore1:%.1f\nscore2:%.1f\n",
            p->num,p->name,p->score[0],p->score[1],p->score[2]);
    return 0;
}
int main( )
{
    struct student stu={12345,"Li Li",67.5,89,78};
    print(&stu);                   // 用结构体指针作函数参数
    return 0;
}
```

5.2 共　用　体

通过使用结构体类型，用户可以方便的根据实际需要来构造新的数据类型，用不同类型的成员存储不同的属性，但结构体的每一个成员均需单独占用一段存储空间。有时，其中的几个成员不会同时出现，每个变量中只会出现一个，为了节省内存空间，能否把这些成员放在同一段内存单元中？

为此，C 语言提供了一种新的构造数据类型——共用体类型（union），它的所有成员共占同一段内存空间，即在某一时刻，只能存储一个成员。

5.2.1 共用体的定义和引用

1．共用体的定义

所谓共用体类型，是指使几个不同类型的变量共同占用同一段内存单元，又称为联合。定义方式如下：

```
union  共用体名
{
    类型标识符 1   成员名 1;
    类型标识符 2   成员名 2;
    …
    类型标识符 n   成员名 n;
}变量表列;
```

例如：

```
union data
{
    int i;
    char ch;
    float f;
}aa,bb,cc;
```

共用体类型也是用户自定义的一种构造类型，与结构体类型一样，也是由不同类型、固定个数的成员组成的。但是结构体的成员分别占用不同的内存单元，是同时存在的；而共用体的所有成员共同占用同一段内存。某一时刻只能有某一个成员存在，其他成员是不存在的。引用共用体变量时，应该注意当前存放在共用体变量中的究竟是哪一个成员。

由于共用体的所有成员共占同一段内存空间，因此整个共用体所占内存空间的大小，等于需要占用内存空间最多的那个成员所占的空间大小。例如，union data 类型的变量 aa，占据的内存空间为 4 个字节，而不是 4+1+4=9 个字节。

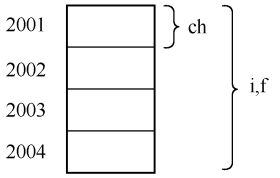

图 5.3　共用体成员在内存中的存储情况

共用体成员在内存中的存储情况如图 5.3 所示。

共用体类型的变量声明形式也有 3 种，同结构体类型。

2．共用体变量的引用

只能引用共用体变量的成员，不能引用共用体变量本身。例如，以下的引用是错误的：

```
printf("%d", aa);
```

正确的引用为：

```
printf("%d\n",aa.i);
printf("%c\n",aa.ch);
aa.f=123.6432;
```

5.2.2　共用体类型的特点

共用体类型的变量有以下特点：

① 每一瞬时只能存放其中一个成员，而不是同时存放几种，即其他成员不起作用。

② 由于共用体的成员占有共同的存储空间，存入新成员后，原来的数据被覆盖，因而共用体中的数据始终是最后一次修改的数据。例如，若依次进行以下赋值：

```
aa.i=1; aa.ch='a'; aa.f=1.5;
```

则执行后起作用的是最后一个赋值语句，只有 aa.f 存在。

③ 共用体变量的地址和它的成员的地址都是同一地址，即&aa 与&aa.i、&aa.ch、&aa.f 都是同一地址。

④ 共用体变量可以初始化，但只能初始化第一个成员。例如：

```
union data
{
    int i;
    char ch;
    float f;
};
union data d1={10};
```

⑤ 共用体变量不能作为函数的参数，也不能作为函数返回值，但可以使用指向共用体变量的指针。

⑥ 共用体类型可以出现在结构体类型中，共用体成员也可以是结构体类型。

⑦ 可以定义共用体数组。

5.2.3　共用体应用举例

【例 5.6】 有一个学校的若干人员的数据，其中有教师、职工和学生。教师的信息包括姓名、年龄、职业、职称，职工的信息包括姓名、年龄、职业、职位，学生的信息包括姓名、年龄、职业、年级。要求输入若干人员的信息并输出。

分析：在每一个结构体变量中，所有数据成员都要单独占用一个存储空间。但是，有些问题中，并不需要所有的数据成员同时出现。例如，在一个学校人员的通用管理程序中，要使用下列数据：

① 姓名（name）

② 年龄（age）

③ 职业（occupation）

④ 级别（rank）：

- 学生（student）：grade（取值：1、2、3、4）
- 教师（teacher）：title（取值：教授、副教授、讲师、助教）
- 职工（worder）：post（校长、处长、科长等）

其中，name、age、occupation 等 3 项对学生和教师职工是一样的，但数据 rank 则因学生、教师或职工而不同。学生的 rank 指年级（grade），教师的 rank 指职称（title），职员的 rank 指职位或职务（post）。因而，grade、title 和 post 不需要同时存储。当 occupation 为 student 时，使用 grade，取值范围为整数 1、2、3、4；当 occupation 为 teacher 时，使用 title，取值为字符串；当 occupation 为 worker 时，使用 post，取值为字符串。也就是说，grade、title 和 post 虽类型不同，但它们不同时出现。出现的时间是根据 occupation 的条件来判断的。

可以用共用体类型来定义学生的年级、教师的职称和职工的职位，在某一时刻，只有一个共用体成员起作用。定义形式如下：

```
union jibie
{
    int grade;
    char title[10];
    char post[16];
};
```

姓名、年龄等信息可以用结构体类型来进行定义。参考程序如下：

```
#include <stdio.h>
#define N 10
union jibie
{
    int grade;              // 学生的年级信息
    char title[10];         // 教师的职称信息
    char post[16];          // 职工的职务信息
};
struct renyuan
{
    char name[20];
    int age;
    char occp;
    union jibie rank;
};
int main( )
{
    struct renyuan person[N];
```

```
    int i;
    for (i=0;i<N;i++)
    {
        printf("Input the %d people\'s name,sex,age,occupation:\n",i+1);
        printf("name:");
        gets(person[i].name);
        printf("age:");
        scanf("%d",&person[i].age);
        getchar( );
        printf("occupation(s or t or w):");
        scanf("%c",&person[i].occp);
        switch(person[i].occp)
        {
            case 's':                          // 职业为学生（s）
                printf("grade:");
                scanf("%d",&person[i].rank.grade);
                break;
            case 't':                          // 职业为教师（t）
                printf("title:");
                scanf("%s",person[i].rank.title);
                break;
            case 'w':                          // 职业为职工（w）
                printf("post:");
                scanf("%s",person[i].rank.post);
                break;
        }
        getchar( );
    }
    for (i=0;i<N;i++)
    {
        printf("name:%s age:%d occupation:%c\n",person[i].name,person[i].age,person[i].occp);
        switch(person[i].occp)
        {
            case 's':printf("grade:%d",person[i].rank.grade);break;
            case 't':printf("title:%s",person[i].rank.title);break;
            case 'w':printf("post:%s",person[i].rank.post);break;
        }
    }
    printf("\n");
    return 0;
}
```

5.3　枚 举 类 型

5.3.1　枚举类型的定义和引用

在实际问题中，有时某些数据的取值用有意义的名字来表示更为直观，例如月份、星期，用相应的英文单词来表示比用整型和字符型数据表示更一目了然。在 C 语言中，用枚举类型定义一些具有赋值范围的变量。枚举类型属于简单基本数据类型，但使用方法与结构体类型相似，因此在本章详细介绍。

1．枚举的概念

所谓“枚举”，是指将变量的所有取值一一列举出来，变量的取值只限于列举出来的值的范围，该变量称为枚举类型变量，所列举的值叫作枚举元素或枚举常量。

2．枚举类型的定义

枚举类型和变量的定义方式如下：

```
enum 枚举类型标识符{枚举元素 1,枚举元素 2,……,枚举元素 n};
enum 枚举类型名 变量列表;
```

例如：

```
enum weekday{sun,mon,tue,wed,thu,fri,sat};
enum weekday day;
```

注意：枚举中每个成员（标识符）分隔符是逗号“,”，不是分号“;”。

3．枚举类型的引用

枚举变量的引用与普通变量是一样的，但要注意枚举变量的取值不能超出所罗列的枚举常量的范围。

例如，有如下定义：

```
enum weekday{sun,mon,tue,wed,thu,fri,sat}workday,weekend,w[3];
```

那么，如下引用是正确的：

```
workday=mon;　或　　w[2]=sat;
```

说明：

① C 语言中枚举元素按常量处理，它们是有值的，它们的值是系统按其定义顺序自动赋值的。在上面的定义中，枚举元素 sun 的值为 0，mon 的值为 1，依次类推。枚举变量的值即它所取的枚举元素的值，此值可用输出语句输出查看。

② 枚举元素的值也可以改变，但必须在定义时指定。例如：

```
enum weekday{sun=7,mon=1,tue,wed,thu,fri,sat};
```

如果定义时未指定值，则按顺序取默认值。

③ 枚举元素是常量，不是变量，不能在定义以外的任何位置对它们赋值，如下的赋值是不正确的：

```
sun=5;
```

④ 枚举值可以用来做判断比较，例如：

```
if (day==sun)　…;
if (day>mon&&day<fri) …
```

5.3.2　枚举类型应用举例

【例 5.7】 程序中提示输入月份数，然后根据输入显示该月的天数。

分析：月份和天数的值都是固定的，可以把这些值用有限个常量来描述。用枚举类型来表示，将变量所能赋的值一一列举出来，给出一个具体的范围。月份的定义如下：

```
enum month{Jan=1,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec};
```

令 1 月份 Jan 的值为 1，后面的值依次增 1，即 Feb 的值为 2，Mar 的值为 3，依次类推。

参考程序如下：

```
#include <stdio.h>
enum month{Jan=1,Feb,Mar,Apr,May,Jun,Jul,Aug,Sep,Oct,Nov,Dec};  // 定义枚举类型
int main( )
{
    enum month m;
    int n;
    printf("          Month and Day\n");
    printf("Please input the month(1-12):");              // 输入要计算的月份
    scanf("%d",&m);
    switch(m)
    {
        case Jan:                                         // 1，3，5，7，8，10，12 月都是 31 天
        case Mar:
```

```
            case May:
            case Jul:
            case Aug:
            case Oct:
            case Dec:
                    n=31;
                    break;
            case Feb:                                   // 不考虑闰年时，2 月为 28 天
                    n=28;
                    break;
            case Apr:                                   // 4，6，9，11 月都是 30 天
            case Jun:
            case Sep:
            case Nov:
                    n=30;
                    break;
            default:
                    printf("Input error!The month must be from 1 to 12!\n");
    return;
        }
        printf("The %d month have %d days\n",m,n);
        return 0;
}
```

从实例中可以看出，使用枚举类型，可让数值与有意义的名字对应，更方便于实际问题的理解。

*5.4　链　　表

5.4.1　概述

链表是一种常见的数据结构，是一种动态地进行存储分配的结构，链表的长度是在程序运行过程中动态确定的。它不同于数组，数组的长度是在定义时确定的。

在实际使用中，如果元素个数不确定（例如，有的班级有 30 名同学，而有的班级有 50 名同学，或者不能确定最多人数），此时定义数组长度必须足够大，而当实际人数很少时，使用该数组，势必造成内存的巨大浪费。

链表是根据实际需要，动态地开辟内存单元，如图 5.4 所示。

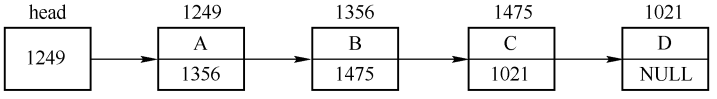

图 5.4　链表

链表中的每一个元素称为结点，每个结点包括两部分：一部分为用户需要用的数据；第二部分为指针，存放下一个结点的地址。

此外，每个链表都有一个头结点 head 和一个尾结点。head 是访问整个链表的开始，指向第 1 个实际结点；尾结点的指针为 NULL，表示结束。

链表中的结点可以不是连续存放的，由上一个元素根据地址找到下一个元素，一环扣一环。

链表结点需要用结构体类型定义，成员中必须有一个指针变量，用来指向下一个结点。例如：

```
struct student
{
```

```
    int num;
    float score;
    struct student *next;
};
```

注意：指针的类型应为结构体类型，因为它要指向下一个结点。

5.4.2　简单链表

简单链表的程序，例如：

```
#include <stdio.h>
struct student
{
    int num;
    float score;
    struct student *next;
};
int main( )
{
    struct student a,b,c,*head,*p;
    a.num=99101;   a.score=89;
    b.num=99102;   b.score=90;
    c.num=99103;   c.score=85;
    head=&a;                                  // 头指针 head 指向第一个结点 a
    a.next=&b;                                // 把结点 b 连在结点 a 的后面
    b.next=&c;                                // 把结点 c 连在结点 b 的后面
    c.next=NULL;                              // 尾结点指针为 NULL
    p=head;
    do
    {
        printf("num: %d score: %5.1f\n",p->num,p->score);
        p=p->next;                            // 用指针变量 p 依次访问各结点
    }while(p!=NULL);
    return 0;
}
```

思考：各个结点如何连成一个链表？p 起什么作用？执行“p=p->next;”后 p 指向谁？

5.4.3　动态链表

5.4.2 节中的结点数都是在程序中定义好的，并不是运行程序后临时开辟的，因此称为静态链表。

动态链表需要特殊的函数，向系统申请存储空间，如 malloc 函数和 calloc 函数，使用完毕后，需使用 free 函数释放申请的空间。

【例 5.8】 编写一个函数建立一个有 3 名学生数据的单向动态链表。

分析：定义 3 个指针变量：head、p1、p2，都是结构体 struct student 类型。用 malloc 函数开辟结点。约定如果输入结点数据的学号为 0，则表示链表建立完毕。head 存放头结点，p1 用于指向新开辟的结点，p2 用于指向已链接的表尾结点。

动态链表的建立过程如下：

```
n=0;
head=NULL;                                          // 头指针设为 NULL
p1=p2=(struct student *)malloc(sizeof(struct student));   // 动态申请内存空间
scanf("%d%f",&p1->num,&p1->score );                 // 如图 5.5（a）所示
while (p1->num!=0)
{
    n=n+1;
    if (n==1) head=p1;                              // 设为头结点
```

```
        else p2->next=p1; // 连到链表末尾
        p2=p1;                  // 修改尾结点
        p1=(struct student *)malloc(sizeof
(struct student));              // 申请新结点
        scanf("%d%f",&p1->num,&p1->score);
    }
    p2->next=NULL;      //尾结点指针赋为 NULL
    free(p1);           //释放最后申请的结点
```

动态链表建立过程演示图如图 5.5 所示。

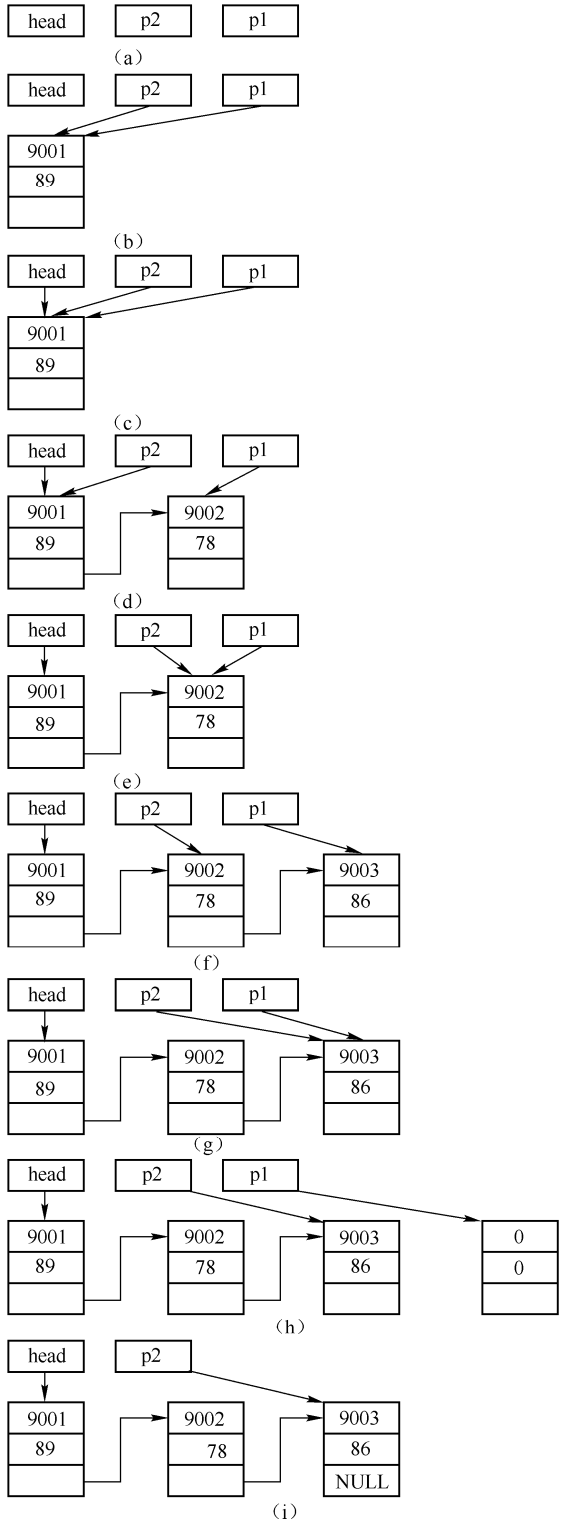

图 5.5　动态链表的建立

5.4.4　链表的实现及应用

输出链表，例如：

```
p=head;
if (head==NULL)
    return;
do
{
    printf("%d%5.1f\n",p->num,p->score);
    p=p->next;
}while(p!=NULL);
```

【例 5.9】　建立链表、搜索链表、插入结点、删除结点等。

参考程序如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <malloc.h>
#include <string.h>
#include <conio.h>
#define N 10
typedef struct node
{
    char name[20];
    struct node *next;
}stud;                  // 建立类型 stud
stud *creat(int n)      // 建立链表
{
    stud *p, *h, *s;
    int i;
    if  ((h=(stud  *)malloc(sizeof(stud)))==
NULL)
    {
        printf("cannot find space!");
        exit(0);
    }
    h->name[0]='\0'; // 初始化头结点
    h->next=NULL;
    p=h;                // p 指向头结点
    for (i=0;i<n;i++)
    {
    if ((s=(stud *)malloc(sizeof(stud)))== NULL)
// 判断内存申请是否成功
        {
            printf("cannot find space!");
            exit(0);
        }
```

```
            p->next=s;                                  // 把 s 连在链表末尾
            printf("please input %d student\'s name:",i+1);
            scanf("%s",s->name);
            s->next=NULL;
            p=s;                                        // 保持 p 总指向链表的最后一个结点
        }
        return(h);
}
stud *search(stud *h,char *x)                           // 在链表中查找 x，返回其所在结点 p
{
        stud *p;
        char *y;
        p=h->next;                                      // p 指向链表的第 1 个结点
        while (p!=NULL)
        {
            y=p->name;
            if (strcmp(y,x)==0)                         // 找到 x
                return(p);                              // 返回结点 p
            else p=p->next;                             // 继续搜索下一个结点
        }
        if (p==NULL)                                    // 若链表结束还没找到
            printf("data not find!");
        return NULL;
}
stud *search2(stud *h,char *x)                          // 在链表中查找 x，返回它的前一个结点 s
{
        stud *p, *s;
        char *y;
        p=h->next;
        s=h;                                            // s 指向 p 的前一个结点
        while (p!=NULL)
        {
            y=p->name;
            if (strcmp(y,x)==0)
                return(s);                              // 返回前一个结点 s
            else
            {
                p=p->next;                              // p 继续后移
                s=s->next;                              // s 继续后移
            }
        }
        if (p==NULL)
            printf("data not find!");
        return NULL;
}
void insert(stud *p)                                    // 在结点 p 后面插入一个新结点
{
        stud *s;
        if ((s=(stud *) malloc(sizeof(stud)))==NULL)    // 申请新结点 s
        {
            printf("cannot find space!");
            exit(0);
        }
        printf("please input the student\'s name:");
        scanf("%s",s->name);
        s->next=p->next;                                // 把 p 的下一个结点连在 s 后面
        p->next=s;                                      // 把 s 连在 p 后面
}
```

```
void del(stud *x,stud *y)                          // 删除结点 x 后面的 y 结点
{
    x->next=y->next;                               // 把 y 的下一个结点连在 x 后面
    free(y);                                       // 释放 y 结点
}
void print(stud *h)                                // 输出链表中
{
    stud *p;
    p=h->next;
    printf("data information:\n");
    while (p!=NULL)
    {
        printf("%s ",p->name);
        p=p->next;
    }
}
void quit( )                                       // 退出程序
{
    exit(0);
}
void menu( )                                       // 程序主界面
{
    printf("\t\t\t simple nextlise realization of c\n");
    printf("\t\t|----------------------------------------|\n");
    printf("\t\t|                                        |\n");
    printf("\t\t| [1]   create nextlist                  |\n");
    printf("\t\t| [2]   seach                            |\n");
    printf("\t\t| [3]   insert                           |\n");
    printf("\t\t| [4]   delete                           |\n");
    printf("\t\t| [5]   print                            |\n");
    printf("\t\t| [6]   exit                             |\n");
    printf("\t\t|                                        |\n");
    printf("\t\t| if no list exist,create first          |\n");
    printf("\t\t|                                        |\n");
    printf("\t\t|----------------------------------------|\n");
    printf("\t\t please input your choose(1-6):");
}
int main( )
{
    int choose;
    stud *head, *searchpoint, *forepoint;
    char fullname[20];
    while(1)
    {
        menu( );                                   // 显示主界面
        scanf("%d",&choose);
        switch(choose)
        {
            case 1:head=creat(N);
                   break;
            case 2:printf("input the student\'s name which you want to find:");
                   scanf("%s",fullname);
                   searchpoint=search(head,fullname);
                   printf("the stud name you want to find is: %s",searchpoint->name);
                   printf("\n push returen to main menu.");
                   getchar( );getchar( );
                   break;
            case 3: printf("input the insert position:");
```

```
                    scanf("%s",fullname);
                    searchpoint=search(head,fullname);
                    printf("the stud name you want to find is: %s",searchpoint->name);
                    insert(searchpoint);
                    print(head);
                    printf("\npush returen to main menu.");
                    getchar( );getchar( );
                    break;
            case 4:print(head);
                    printf("\ninput the student\'s name which you want to delete:");
                    scanf("%s",fullname);
                    searchpoint=search(head,fullname);
                    forepoint=search2(head,fullname);
                    del(forepoint,searchpoint);
                    break;
            case 5:print(head);
                    printf("\npush returen to main menu.");
                    getchar( );getchar( );
                    break;
            case 6:quit( );
                    break;
            default:
                        printf("illegal letter!push returen to main menu.");
                        menu( );
                        getchar( );
        }
    }
    return 0;
}
```

习　　题

一、选择题

1．设有以下说明语句，则以下叙述不正确的是（　　）。

```
struct stu
{
    int a;
    float b;
}stutype;
```

A）struct 是结构体类型的关键字　　B）struct stu 是用户定义的结构体类型

C）stutype 是用户定义的结构体类型名　　D）a 和 b 都是结构体成员名

2．当说明一个结构体变量时系统分配给它的内存空间是（　　）。

A）各成员所需内存空间的总和　　B）结构中第一个成员所需内存空间

C）成员中占内存空间最大者所需的容量　　D）结构中最后一个成员所需内存空间

3．以下对结构体变量 stu1 中成员 age 的非法引用是（　　）。

```
struct student
{
    int age;
    int num;
}stu1,*p;
p=&stu1;
```

A）stu1.age　　B）student.age　　C）p->age　　D）(*p).age

4．当说明一个共用体变量时系统分配给它的内存是（　　）。

A）各成员所需内存量的总和　　B）结构中第一个成员所需内存量

C）成员中占内存量最大者所需的容量　　D）结构中最后一个成员所需内存量

5．若有定义“union{char a[10]; int b; char c[20];}comm;”，当执行以下语句，输出结果是（　　）。

```
strcpy(comm.a,"hello");
comm.b=3;
strcpy(comm.c,"my computer");
printf("%s",comm.a);
```

A）hello　　B）hello my computer　　C）my computer　　D）hellomputer

6．设有如下枚举类型定义，则枚举常量 Italian 的值为（　　）。

```
enum language{English=6,French,Chinese=1,Japanese,Italian};
```

A）10　　B）4　　C）3　　D）5

7．以下对枚举类型名 ee 的定义中正确的是（　　）。

A）enum ee{A,B,C,D};　　B）enum ee{'A','B','C','D'};

C）enum ee={A,B,C,D};　　D）enum ee={'A','B','C','D'};

8．已知学生记录描述为：

```
struct student
{
    int no;
    char name[20],sex;
    struct
    {
        int year,month,day;
    }birth;
};
struct student s;
```

设变量 s 中的“生日”为 1999 年 11 月 12 日，则能正确赋值的程序段是（　　）。

A）year=1999;　month=11;　day=12;

B）s.year=1999;　s.month=11;　s.day=12;

C）birth.year=1999;　birth.month=11;　birth.day=12;

D）s.birth.year=1999;　s.birth.month=11;　s.birth.day=12;

二、填空题

1．同数组类似，一个结构体也是若干数据项的集合，但与数组不同，数组中的所有元素_____，而结构体中的数据项_____。

2．使用结构体处理数据的场合是_____。

3．_____运算符用于引用结构体的成员，_____运算符用于引用结构体指针所指对象的成员。

4．在任一时刻，共用体中能存放_____个有效的成员数据。

5．枚举中的元素不是变量，也不是字符串，它是_____。

6．有枚举类型定义如“enum T{a1, a2=4, a3, a4=10};”，则枚举常量 a1 的值为_____，a3 的值为_____。

三、程序填空题

1．以下程序输出结构体类型数组初值中的字符串“Mary”。

```
#include <stdio.h>
```

```
int main( )
{
    struct person
    {
        char name[20];
        int age;
    };
    struct person class[10]={{"John",18},{"Paul",20},{"Mary",18},{"Rose",20}};
    printf("%s\n",_________);
    return 0;
}
```

2．以下程序运行的结果是 10,x，请填空。

```
#include <stdio.h>
struct n
{
    int x;
    char c;
};
int main( )
{
    struet n a={10,'x'};
    printf("%d,%c\n",_________,_________);
}
```

3．以下程序的运行结果是_____。

```
#include <stdio.h>
struct
{
    int a;
    int b;
    struct
    {
        int x;
        int y;
    }ins;
}outs;
int main( )
{
    outs.a=11;
    outs.b=4;
    outs.ins.x=outs.a+outs.b;
    outs.ins.y=outs.a-outs.b;
    printf("%d,%d\n",outs.ins.x,outs.ins.y);
    return 0;
}
```

4．以下语句定义的结构体类型里包含两个成员，其中成员变量 info 用来存入整型数据，成员变量 link 是指向自身结构体的指针，请将定义补充完整。

```
struct node
{
    int info;
    _________ link;
};
```

5．执行下列程序，程序的输出结果为_____。

```
#include <stdio.h>
int main( )
{
    union ts
```

```
    {
        int i;
        char c[2];
    }x;
    x.c[0]='\0';
    x.c[1]='A';
    printf("%d\n",x.i);
    return 0;
}
```

四、编程题

1．编写函数 readinfo 读入 5 名学生的编号（整型）、姓名（字符串）、3 门课程的成绩（实型数组）放在结构体数组 student 中。编写函数 writeinfo 输出 10 名学生的记录，在主函数中分别调用上述两个函数，实现程序的功能。

2．定义一个结构体用于存储年、月、日数据，输入某一年中的任意两个日期；定义一个函数求两个日期之间的天数，要求在主函数中输入日期，并输出结果。

3．已知某公司设有 5 种职务，职务和基本工资的对应关系见表 5.3。请输入某个职务的编号，输出其基本工资。

4．利用指向结构体的指针变量处理 5 名学生的信息（学号、姓名和一门课程的成绩）。要求定义一个结构体数组，在主函数中从键盘输入 5 名学生的信息。编写自定义函数 max 求 5 名学生中成绩最高的学生的学号，并在主函数中输出该学生的所有信息。

表 5.3

编　　号	职　　务	基 本 工 资
1	General Manager	5000
2	Marketing Manager	4000
3	Project Manager	3500
4	Service Manager	3000
5	Staff	2000

第 6 章　磁盘数据存储

经过前几章的学习，已经可以使用数组、结构体等构造数据类型进行复杂程序的编写，且能将任务按功能模块分解，进行较大型的程序开发。

实际程序设计中，还会遇到一个问题，如果程序处理的数据需要长期保存起来该怎么办？虽然数组可以用于存储多条信息，但数组属于程序运行过程中的临时变量，当程序运行结束后系统将释放其所占的内存空间，处理结果就不再被保存了。另外，有时程序处理的数据不是从键盘上输入的，而是从数据文件里读取出来的。这又将如何实现呢？这就是本章所要解决的问题。

本章介绍：C 语言中文件系统的相关知识，重点介绍使用文件存取数据的步骤及打开文件函数的使用方法。针对不同类型和不同使用目的的数据，介绍各种分类读写文件的函数。此外，还将介绍与文件操作相关的一些函数，如文件位置指针控制函数和读写检测函数等。

6.1　将数据写入文件

【例 6.1】 将一串字符'A'～'Z'写入文件保存起来。

```
#include <stdio.h>
#include <stdlib.h>                          // 此头文件中包含 exit 函数的说明
int main( )
{
    char ch;
    FILE *fp;                                // 步骤①，定义文件指针
    fp=fopen("letter.txt","w");              // 步骤②，新建并打开一个磁盘文件
    if (fp==NULL)                            // 判断文件打开是否成功
    {
        printf("Cannot open file letter.txt!\n");
        exit(0);                             // 提前结束程序
    }
    for (ch='A';ch<='Z';ch++)
        fputc(ch,fp);                        // 步骤③，读写文件；此处为写入字符
    fputc('\n',fp);                          // 最后写入一个换行符
    fclose(fp);                              // 步骤④，关闭文件
    return 0;
}
```

说明：

由以上简单示例可以看出使用文件的一些必要步骤和操作特点：

① 定义文件类型指针。文件类型 FILE 是结构体类型，在头文件 stdio.h 中已经声明，只需包含该头文件，即可使用 FILE 类型。C 语言使用的是缓冲文件系统，系统自动在内存中为每一个正在使用的文件开辟一个缓冲区。文件指针指向这一内存区的首地址。输出时，程序中的数据先送入缓冲区，装满后或当文件关闭时才一起输出到磁盘；输入时，先从磁盘读一批数据到缓冲区，然后再将数据送到程序中。

② 打开文件。使用文件，不论是写入数据还是读出数据，不论是对一个已有的文件进行读写，还是对新文件进行操作，首先都要用 fopen 函数打开文件。

③ 向文件写入数据，或从文件读取数据。写入和读取的数据方式不同，需要使用不同的读写文件函数，这是关键的一步。

④ 关闭文件。这是文件使用的最后一步，使用完毕必须关闭文件，才能彻底的将文件缓冲区的数据写入文件，并释放系统分配的文件缓冲区。

以上是磁盘数据存储，即文件操作的几个通用步骤，当然实际的文件操作不限于这几个步骤。例如打开文件后可以检测打开是否成功等，再如文件使用过程中使文件指针复位或调整等，也都是常用操作，但文件操作至少要包含上述 4 个基本步骤。

6.1.1　打开文件函数

格式：

```
FILE *fopen("文件名","打开方式");
```

功能：打开指定名称的文件。

返回：如果成功，则返回这一文件缓冲区的首地址，否则返回空指针 NULL。

说明：文件名是指包含主文件名和扩展名的文件全称，如果文件不在当前目录下，还应该给出文件的路径。例如:

```
fp=fopen("c:\\temp\\file1.txt1","r");
```

C 语言操作的数据文件有两种格式：二进制文件和 ASCII 码文件。其存储形式有所不同，ASCII 码文件也叫文本文件，每个字节存放一个 ASCII 码，代表一个字符，因而便于对字符进行逐个处理（如字符串）。ASCII 码文件可以阅读、可以打印，但它与内存数据交换时需要转换。二进制文件将内存中的数据按照其在内存中的存储形式原样输出，并保存在文件中。内存数据和磁盘数据交换时无须转换，但是二进制文件不可直接阅读、打印。它的 1 个字节并不对应 1 个字符。以整数占 2 个字节为例，数值 32767 在内存中的二进制数形式为 01111111 11111111，写入到二进制文件中将占 2 个字节，存储形式就是它的内存二进制数形式。而如果用一个文本文件存储 32767，将作为 5 个数字字符存储，因此文件将占 5 个字节的长度。

两种不同格式的文件在打开方式上的区别是：二进制文件在原打开方式后面加一个“b”，来表示操纵的是二进制文件。例如，以读数据的方式打开文本文件，打开方式用“r”，而打开二进制文件读取数据，打开方式应为“rb”。具体的几种打开方式如下：

① 读的方式打开文件。使用“r”和“r+”（r+表示既可读文件中的数据，也可将数据写入文件），使用 r 的方式文件应该是已经存在或建立过的文件。

② 写的方式打开文件。使用“w”和“w+”（w+表示既可读文件中的数据，也可将数据写入文件）。使用 w 的方式表示新建一个文件，若磁盘上已有同名文件，则被覆盖。

③ 追加的方式打开文件。使用“a”和“a+”时，文件如果已经存在，原有内容不被删除，文件打开后，文件位置指针直接移到数据末尾，在后面追加内容。文件如果不存在，则新建一个文件，写入数据。

6.1.2　关闭文件函数

格式：

```
int fclose(FILE *fp);
```

功能：关闭 fp 指向的文件。

返回：如果文件正常关闭将返回 0 值，如果关闭文件失败则返回 EOF(−1)。

以各种使用方式打开的文件，其关闭函数是相同的。函数的参数为打开该文件时返回的文件指针。

【例 6.2】 将图书信息数据存储于数据文件中，观察该例所使用的写入文件函数。

分析：第 5 章介绍了结构体类型，并在实验中创建了图书管理系统的实例。在本章中可以将实例中的图书信息写入磁盘文件中保存起来。考虑到图书信息为复杂构造数据类型并且数据量比较大，在

写入图书信息时可以使用二进制的块写函数 fwrite，将图书信息一次成批地写入文件中。这样写入效率高，文件长度小。本例中并不讲解 fwrite 函数的使用细节，主要是让大家观察不同的写入信息可以采取不同的读写函数，但文件操作的几个通用步骤是大致相同的。

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define N 5
int main( )
{
    struct book
    {
        int num;                    // 图书编号
        char name[50];              // 图书名
        char publish[40];           // 出版社
        struct
        {
            int year;
            int month;
        }date;                      // 出版时间
        char borrow;                // 借阅标志，表示是否已被借阅
    };
    struct book books[N];           // 定义图书类型的数组，用于临时存储 N 本图书的信息
    char name[50];
    int i,num;
    FILE *fpbook;                   // 定义文件指针
    fpbook=fopen("books.dat","wb+");                    // 新建一个二进制文件 books.dat
    if (fpbook==NULL)                                   // 判断文件打开成功与否
    {
        printf("Cannot open file books.dat!\n");
        exit(0);
    }
    // 提示输入图书的信息
    printf("Input book\'s num,name,publish,date,borrow:\n");
    printf("Be caution: you should input the <Space> to seperate every item!\n");

    for (i=0;i<N;i++)                                   // 输入 N 本图书的信息
    {
        printf("number %d:",i+1);
        scanf("%d",&books[i].num);
        scanf("%s %s",books[i].name,books[i].publish);
        scanf("%d%d%c",&books[i].date.year,&books[i].date.month,&books[i].borrow);
    }
    // 将图书信息写入到文件 books.dat 中
    fwrite(books,sizeof(struct book),N,fpbook);         // 此处使用 fwrite 函数写入文件
    fclose(fpbook);                                     // 关闭文件
    return 0;
}
```

说明：仔细观察，在文件使用方面，打开函数 FOPEN 的使用略有不同，而且向文件写入数据也使用了不同的函数。

6.2　文件读写分类函数

由例 6.2 可以看出，对于不同文件格式或数据类型可以使用不同的读写函数。例 6.2 中的图书信息使用 fwrite 函数将信息以二进制数形式一次性写入文件，而例 6.1 中则使用 fputc 函数将字符一个一个

地写入文件中。下面来总结和归纳各种类型的文件读写函数。

6.2.1　单字符写入函数

格式：

int fputc(int ch, FILE *fp);

功能：将字符（ch 可以是字符表达式、字符常量、字符变量等）写入 fp 所指向的文件。

返回：输出成功，返回输出字符的 ASCII 码值；输出失败，返回 EOF（该符号常量在 stdio.h 的头文件中声明，值为–1）。

其他说明：每次写入一个字符，文件位置指针自动指向下一个字节。

【例 6.3】 从键盘输入一行字符，写入到文本文件 string.txt 中。

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    FILE *fp;
    char ch;
    if ((fp=fopen("string.txt","w"))==NULL) // 以写方式打开新文件 string.txt
    {
        printf("Cannot open file string.txt!\n");
        exit(0);
    }
    do                                      // 不断从键盘读字符并写入文件，直到遇到换行符
    {
        ch=getchar( );                      // 从键盘读取字符
        fputc(ch,fp);                       // 将字符写入文件
    }while(ch!='\n');
    fclose(fp);                             // 关闭文件
    return 0;
}
```

6.2.2　单字符读取函数

格式：

int fgetc(FILE *fp);

功能：从 fp 所指向的文件中读出一个字符，字符由函数返回。返回的字符可以赋值给字符变量，也可以直接参与表达式运算。

返回：读取成功返回读取的字符，遇到文件结束，返回 EOF（–1）。

其他说明：每次读出一个字符，文件位置指针自动指向下一个字节。

文本文件的内部全部是 ASCII 字符，其值不可能是–1，所以可以使用 EOF 确定文件是否结束。但是对于二进制文件不能这样做，因为可能在文件中间某个字节的值恰好等于–1，此时使用–1 来判断文件结束是不恰当的。为了解决这个问题，ANSI C 提供了 feof 函数判断文件是否真正结束。

feof 函数既适用于文本文件，也适用于二进制文件文件结束的判断。

【例 6.4】 将磁盘上一个文本文件的内容复制到另一个文件中。

参考程序如下：

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    FILE *fp_in, *fp_out;
    char infile[20],outfile[20];
```

```
    printf("Input the infile name:\n");
    scanf("%s",infile);                          // 输入要复制的源文件的文件名
    printf("Input the outfile name:\n");
    scanf("%s",outfile);                         // 输入复制目标文件的文件名
    if ((fp_in=fopen(infile,"r"))==NULL)         // 打开源文件
    {
        printf("Cannot open file: %s!\n",infile);
        exit(0);
    }
    if ((fp_out=fopen(outfile,"w"))==NULL)       // 打开目标文件
    {
        printf("Cannot open file: %s\n!",outfile);
        exit(0);
    }
    while (!feof(fp_in))                         // 若源文件未结束，则继续读写文件
    {
        fputc(fgetc(fp_in),fp_out);              // 从源文件读一个字符，写入目标文件
    }
    fclose(fp_in);                               // 关闭源文件
    fclose(fp_out);                              // 关闭目标文件
    return 0;
}
```

6.2.3　字符串读取函数

格式：

```
char *fgets(char *str,int n,FILE *fp);
```

功能：从 fp 所指向的文件读 *n*–1 个字符，并将这些字符放到以 str 为起始地址的内存单元中。如果在读入 *n*–1 个字符结束前遇到换行符或 EOF，读入结束。字符串读入后最后加一个'\0' 字符。

返回：读入成功返回读入串的首地址，遇到文件结束或出错返回 NULL。

【例 6.5】 编制一个将文本文件中全部信息显示到屏幕上的程序（类似于 dos 的 type 命令）。

参考程序如下：

```
#include <stdio.h>
#include <stdlib.h>
int main(int argc,char *argv[ ])
{
    FILE *fp;
    char string[81];          // 最多保存 80 个字符，外加一个字符串结束标志'\0'
    if (argc!=2||(fp=fopen(argv[1],"r"))==NULL)   // 打开从命令行指定的文件
    {
        printf("Cannot open file %s\n!",argv[1]);
        exit(0);
    }
    // 如果未读到文件末尾（EOF），则继续循环
    while (fgets(string,81,fp)!=NULL)             // 从文件最多读 80 个字符
        printf("%s\n",string);                    // 输出读到的字符串
    fclose(fp);                                   // 关闭文件
    return 0;
}
```

说明：此例中 main 为带参主函数。程序文件编译、连接后生成扩展名为.exe 的可执行文件。运行程序时，要在 DOS 命令窗口中的命令提示符后，输入程序文件名、空格，再输入参数。例如，程序文件名为 f1，则输入 f1 myfile.txt，此例中参数的取值情况为 argc=2，表明有包括程序文件名在内的两

个字符串，argv[]数组存放上述两个字符串，即 argu[0]存放 f1，argv[1]存放 myfile.txt。因此，本例中打开文件函数 fopen 使用的参数为 argv[1]。

6.2.4　字符串写入函数

格式：

```
fputs(char *str,FILE *fp);
```

功能：向 fp 所指向的文件中写入以 str 为首地址的字符串。

返回：写入成功返回值 0，出错返回非 0 值。

【例 6.6】 在文本文件 string.txt 末尾添加若干行字符。

参考程序如下：

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main( )
{
    FILE *fp;
    char str[81];
    if ((fp=fopen("string.txt","a"))==NULL)          // 以追加方式打开文件
    {
        printf("Cannot open file string.txt!\n");
        exit(0);
    }
    while (strlen(gets(str))>0)     // 从键盘读入一个字符串，遇到空行（长度为 0）则结束
    {
        fputs(str,fp);                               // 将字符串写入文件中
        fputs("\n",fp);                              // 输出字符串后换行
    }
    fclose(fp);                                      // 关闭文件
    return 0;
}
```

6.2.5　格式化读写函数

格式化文件读写函数 fprintf、fscanf 与函数 printf、scanf 作用基本相同，区别在于 fprintf、fscanf 读写的对象是磁盘文件，而 printf、scanf 读写的对象是标准输入/输出终端。

格式：

```
int fprintf(FILE *fp,格式字符串,输出列表);
int fscanf(FILE *fp,格式字符串,输入地址列表);
```

其中，fp 是文件指针。

功能：fprintf 函数按照格式字符串指定的格式将输出列表的数据写入到磁盘文件，fscanf 函数按照格式字符串指定的格式从磁盘文件读入数据，并存入对应的地址列表空间里。

返回值：返回实际输出或输入的数据个数。

【例 6.7】 从键盘输入学生的姓名和成绩，保存在文本文件 score.txt 中。

参考程序如下：

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main( )
{
    FILE *in_fp;
    char name[20];
```

```
    float score;
    if ((in_fp=fopen("score.txt","w"))==NULL)     // 以只写方式打开新的文本文件
    {
        printf("Cannot creat file score.txt!\n");
        exit(0);
    }
    printf("Input name: ");
    gets(name);
    printf("Input score: ");
    scanf("%f",&score);
    fprintf(in_fp,"%s\n%.1f\n",name,score);          // 把学生姓名和成绩写入文件
    fclose(in_fp);
    return 0;
}
```

【例 6.8】 打开上例中的文本文件 score.txt，读出学生的姓名和成绩输出。

参考程序如下：

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
int main( )
{
    FILE *out_fp;
    char name[20];
    float score;
    if ((out_fp=fopen("score.txt","r"))==NULL)     // 以只读方式打开文件
    {
        printf("Cannot open file score.txt!\n");
        exit(0);
    }
    fscanf(out_fp,"%s%f",name,&score);             // 读入数据时，中间不要加附加格式符
    printf("Name: %s\n",name);
    printf("Score: %.1f\n",score);
    fclose(out_fp);
    return 0;
}
```

【例6.9】 用格式化文件读写函数编写一个程序，该程序用于测试如下自定义函数 function 是否正确。提示磁盘上已经存在一个测试数据文件 prog.in，本题要求从 prog.in 中读出测试数据来测试函数 function，并把函数调用后的数据存入文件 prog.stu 中。

函数 function 信息如下：

```
void function(int *p1,int *p2)
{
    int p;
    p=*p1; *p1=*p2; *p2=p;
}
```

prog.in 文件内容如图 6.1，假设测试数据为 6 组。

1	2
4	5
–9	8
4	2
–5	–1
0	4

图 6.1　prog.in 文件内容

参考程序如下：

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
void function(int *,int *);
int main( )
{
    FILE *fpin,*fpout;
    int i,tnum;
```

```
    fpin=fopen("prog.in","rb");                    // 以只读方式打开新的二进制文件
    if (!fpin)
    {
        puts("File prog.in read error! Call teacher!\n");
        exit(0);
    }
    fscanf(fpin,"%d",&tnum);                       // 从文件中读出数据的组数
    fpout=fopen("prog.stu","wb");
    if (!fpout)
    {
        puts("File prog.stu write error! Call teacher!\n");
        exit(0);
    }
    fprintf(fpout,"%d\n",tnum);                    // 将数据组数写入文件
    for (i=0;i<tnum;i++)
    {
        int a,b;
        fscanf(fpin,"%d%d",&a,&b);                 // 从文件中读出一组数据
        function(&a,&b);                           // 调用测试函数
        fprintf(fpout,"%d%d\n",a,b);               // 将该组数据写入文件
    }
    fclose(fpin);
    fclose(fpout);
    return 0;
}
```

如果程序 function 正确，将会生成 prog.stu 文件，该文件内容如图 6.2 所示。

```
2    1
5    4
8   –9
2    4
–1  –5
4    0
```

图 6.2　prog.stu 文件内容

6.2.6　数据块读写函数

从文件（通常是二进制文件）读写一整块数据（如一个数组、一个结构体变量的数据记录），使用数据块读写函数非常方便，可以实现对数据的整体读入/读出，提高数据存取的效率。

格式：

```
int fread(void *buffer,int size,int count,FILE *fp);
int fwrite(void *buffer,int size,int count,FILE *fp);
```

其中，buffer 是指针，对 fread 函数而言，buffer 指针用于存放从磁盘读入数据块的内存首地址，通常是与读入数据类型相同大小相同的数组名；对 fwrite 函数而言，buffer 指针是指要输出数据的内存首地址，将该起始地址的数据整块的写入磁盘文件中。size 是一个数据块的字节数（每块大小）。count 是要读写的数据块个数。

功能：fread 函数从磁盘文件中读入 size*count 字节数大小的数据，放入到 buffer 指针指向的内存空间。fwrite 函数将 buffer 指针开始，大小为 size*count 字节数的内存空间数据写入到磁盘文件中。

返回值：fread、fwrite 函数返回读取/写入的数据块块数（正常情况为 count）。

注意：以数据块方式读写，文件通常以二进制方式打开。例如，例 6.2 中的 *N* 本图书信息整体写入文件中保存起来。

【例 6.10】 从键盘输入一批学生的数据，然后把它们转存到磁盘文件 stud.dat 中。

参考程序如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
struct student                                  // 共 5 个成员，占用 33Bytes
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
};
int main( )
{
    FILE *fp;
    struct student stud;
    char numstr[20],ch;                         // numstr 用于临时保存学号/年龄/成绩
    if ((fp=fopen("stud.dat","wb"))==NULL)      // 以写方式打开二进制文件
    {
        printf("Cannot open file stud.dat!\n");
        exit(0);
    }
    do
    {
        printf("Input number: ");
        gets(numstr);
        stud.num=atoi(numstr);                  // atoi 函数用于把字符串转换成整型数据
        printf("Input name: ");
        gets(stud.name);
        printf("Input sex: ");
        stud.sex=getchar( );
        getchar( );                             // 略过输入性别后的空格或回车符
        printf("Input age: ");
        gets(numstr);
        stud.age=atoi(numstr);
        printf("Input score: ");
        gets(numstr);
        stud.score=atof(numstr);                // 把字符串转换成浮点数
        fwrite(&stud,sizeof(struct student),1,fp);  // 将该学生信息写入文件
        printf("have another student record(Y/N)? ");
        ch=getchar( );
        getchar( );                             // 读走输入字符后面的回车符
    }while(toupper(ch)=='Y');                   // 继续循环，以读、写下一个学生的数据
    fclose(fp);
    return 0;
}
```

说明：在输入字符，并按回车键后，实际缓冲中有两个字符（如'f'或'm'和'\n'），只需要前面有意义的字符（'f'或'm'）。这时可以用 getchar 函数“空读”略过'\n'。什么情况下要空读？如果后面读取键盘数据是读取数字（整数/浮点数），不必空读；如果后面读取键盘数据是读字符或字符串，应当加一个“空读”，如例 6.10 中的 getchar 函数。

思考：如果输入两个记录的数据，那么最后产生的文件长度是多少？（58 B）

6.3　文件定位函数

对文件的读写可以顺序读写，也可以随机读写。文件顺序读写是从文件的开头开始，依次读写数据（从文件开头读写直到文件尾部）。文件随机读写（文件定位读写）是从文件的指定位置读写数据。

在文件的读写过程中，文件位置指针指出了文件的当前读写位置（实际上是下一步读写位置），每次读写后，文件位置指针自动更新指向新的读写位置（实际上是下一步读写位置）。可以通过文件位置指针函数，实现文件的定位读写。

注意：区分文件位置指针和文件指针。

6.3.1　位置指针复位函数

格式：

```
void rewind(FILE *fp);
```

功能：将文件指针重新定位到文件开始的地方。

返回值：此函数没有返回值。

【例 6.11】 打开例 6.10 中建立的学生数据文件 stud.dat，读出学生信息输出到屏幕上，再读一遍，将其中的学号、姓名、成绩信息另存到 stuscore.dat 文件中。

参考程序如下：

```
#include <stdio.h>
#include <stdlib.h>
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
};
int main( )
{
    struct student stud;
    FILE *fp1,*fp2;
    if ((fp1=fopen("stud.dat","rb"))==NULL)              // 以读方式打开二进制文件
    {
        printf("Cannot open file stud.dat!\n");
        exit(0);
    }
    while (!feof(fp1))                                  // 检测文件读写是否到达文件末尾
    {
        fread(&stud,sizeof(struct student),1,fp1);      // 读一个学生的数据
        printf("student information is:\n");
        printf("number: %d;   name: %s;    sex: %c;    age: %d; score: %.1f;\n",
            stud.num,stud.name,stud.sex,stud.age,stud.score);
    }
    rewind(fp1);                                        // 将文件位置指针重新指向文件开始处
    if ((fp2=fopen("stuscore.dat","w"))==NULL)          // 以写方式建立文本文件
    {
        printf("Cannot open file stuscore.dat!\n");
        exit(0);
    }
    while (!feof(fp1))                                  // 循环读、写数据
```

```
    {
        fread(&stud,sizeof(struct student),1,fp1);        // 读一个学生的数据
        fprintf(fp2,"%d,%s,%.1f\n",stud.num,stud.name,stud.score);
    }
    fclose(fp1);                                          // 关闭文件
    fclose(fp2);
    return 0;
}
```

【例 6.12】 继续例 6.2，从图书信息数据文件中读取图书信息，并修改和保存图书信息到数据文件中。

思考：此种情况下应采用何种打开方式？使用什么样的读写函数对图书信息数据文件中的图书信息数据进行读取？

参考程序如下：

```
#include <stdio.h>
#include <string.h>
#include <stdlib.h>
#define N 5
struct book
{
    int num;                          // 图书编号
    char name[50];                    // 图书名
    char publish[40];                 // 出版社
    struct
    {
        int year;
        int month;
    }date;                            // 出版时间
    char borrow;                      // 是否已被借阅
};
void output(struct book *b);
void borrow(struct book *b,char name[ ]);
void back(struct book *b,char *name);
int main( )
{
    struct book books[N];
    char name[50];
    FILE *fp;
    fp=fopen("books.dat","rb+");      // 以读、写方式打开例 6.2 建立的二进制文件
    if (fp==NULL)                     // 判断文件是否打开成功
    {
        printf("Cannot open file books.dat!\n");
        exit(0);
    }
    fread(books,sizeof(struct book),N,fp);          // 读出 N 本图书的信息
    printf("Now you can input the book name you are going to borrow:\n");
    scanf("%s",name);                               // 读入要借图书的书名
    borrow(books,name);                             // 调用借书函数
    rewind(fp);                                     // 将文件指针重新定位到文件头
    fwrite(books,sizeof(struct book),N,fp);         // 写入修改后的图书信息
    output(books);                                  // 调用屏幕显示函数
    printf("Now you can input the book name you are going to return:\n");
    scanf("%s",name);
    back(books,name);                               // 调用还书函数
    rewind(fp);
    fwrite(books,sizeof(struct book),N,fp);
    output(books);
```

```
    fclose(fp);
    return 0;
}
void output(struct book *b)                          // 图书信息输出函数
{
    int i;
    printf("\nOutput the book\'s information:\n");
    printf("----------------------------------\n");
    printf(" num name publish publishdate borrow:\n\n");
    for(i=0;i<N;i++,b++)
        printf("%–10d%–16s%–12s%6d, %d%9c\n",b–>num,b–>name,b–>publish,
            b–>date.year,b–>date.month,b–>borrow);
    printf("-------------------------------------\n\n");
}
void borrow(struct book *b, char name[ ])            // 图书借阅函数
{
    int i,flag=0;
    for (i=0;i<N;i++,b++)                            // 查找是否有要借的书
    {
        if (strcmp(b–>name,name)==0)                 // 找到要借的书
        {
            flag=1;                                  // 设置找到标志
            if (b–>borrow=='n'||b–>borrow=='N')
            {
                b–>borrow='Y';                       // 设置借阅标志为“已借出”
                printf("Success!\n");
            }
            else                                     // 提示图书已借出
                printf("The book has been borrowed.You Cannot borrow it.\n");
        }
    }
    if (flag==0)                                     // 未找到该图书
        printf("We Cannot find this book.\n");
}
void back(struct book *b, char *name)                // 还书函数
{
    int i,flag=0;
    for (i=0;i<N;i++,b++)                            // 查找要还的书
    {
        if (strcmp(b–>name,name)==0)                 // 找到要还的书
        {
            flag=1;
            b–>borrow='N';                           // 设置未借阅标志
            printf("The book has been returned.\n");
        }
    }
    if (flag==0)
        printf("Cannot find this book! The book doesn\'t belong to our library.\n");
}
```

6.3.2　位置指针的随机移动函数

格式：

```
int fseek(FILE *fp, long offset, int base);
```

功能：将文件的位置指针移到以 base 为起始点，以 offset 为位移量的位置，同时清除文件结束标志。

返回值：函数调用成功返回当前位置，否则返回–1。

base 常用的 3 个符号常量为 SEEK_SET、SEEK_CUR 和 SEEK_END，分别表示文件开始位置、

当前位置和文件末尾位置, 在头文件 stdio.h 中已定义，值分别为 0、1、2。

offset 表示以上述 3 个位置为起始点，文件指针向前或向后移动的字节数。offset 值可正、可负，正值表示向文件末尾的方向移动指针，负值表示向文件头的方向移动指针。

【例 6.13】 打开例 6.10 中建立的学生数据文件 stud.dat，将编号为 2、4 的学生信息输出到屏幕上。

参考程序如下：

```
#include <stdio.h>
#include <stdlib.h>
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
};
int main( )
{
    struct student stud;
    FILE *fp;
    if ((fp=fopen("stud.dat","rb"))==NULL)                    // 以读方式打开二进制文件
    {
        printf("Cannot open file stud.dat!\n");
        exit(1);
    }
    if (fseek(fp,sizeof(struct student),SEEK_SET)== –1)       // 移动文件位置指针到第 2 个学生处
        printf("Error!\n");
    else
    {
        fread(&stud,sizeof(struct student),1,fp);             // 读出第 2 个学生的信息
        printf("student information is:\n");                  // 输出第 2 个学生的信息
        printf("number: %d;name: %s;sex: %c;age: %d;score: %.1f;\n",
                stud.num,stud.name,stud.sex,stud.age,stud.score);
    }
    if (fseek(fp,sizeof(struct student),SEEK_CUR)== –1)       // 移动文件位置指针到第 4 个学生处
        printf("Error!\n");
    else
    {
        fread(&stud,sizeof(struct student),1,fp);             // 读出第 4 个学生的信息
        printf("student information is:\n");                  // 输出第 4 个学生的信息
        printf("number: %d;name: %s;sex: %c;age: %d;score: %.1f;\n",
                stud.num,stud.name,stud.sex,stud.age,stud.score);
    }
    fclose(fp);                                               // 关闭文件
    return 0;
}
```

6.3.3　文件指针当前位置函数

格式：

```
long ftell(FILE *fp);
```

功能：返回 fp 所指向的文件中的读写位置。

返回值：返回位置指针相对于文件开头的位移量。若出错，则返回-1L。

【例 6.14】 打开例 6.10 中建立的学生数据文件 stud.dat，将编号为 3 的学生信息输出到屏幕上，同时显示当前文件指针位置。

参考程序如下：

```
#include <stdio.h>
#include <stdlib.h>
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
};
int main( )
{
    struct student stud;
    FILE *fp;
    long int where;
    if ((fp=fopen("stud.dat","rb"))==NULL)                    // 以读方式打开二进制文件
    {
        printf("Cannot open file stud.dat!\n");
        exit(0);
    }
    if (fseek(fp,2*sizeof(struct student),SEEK_SET)== –1)    // 移动文件位置指针到第 3 个学生处
        printf("Error!\n");
    else
    {
        fread(&stud,sizeof(struct student),1,fp);            // 读出第 3 个学生的信息
        printf("student information is:\n");
        printf("number: %d;  name: %s;  sex: %c;  age: %d;  score: %.1f;\n",
                stud.num,stud.name,stud.sex,stud.age,stud.score);
    }
    where=ftell(fp);
    printf("Current file pointer is %ld\n",where);
    fclose(fp);                                              // 关闭文件
    return 0;
}
```

此例运行后显示的文件位置指针值为 108 个字节（结构体 struct student 类型的长度为 36）。

6.4　其他文件函数

6.4.1　文件结束检测函数

格式：

```
int feof(fp);
```

功能：检测文件指针是否到达文件末尾，即文件是否结束。

返回值：遇到文件结束符返回非零值，否则返回 0。

例如例 6. 11 中的程序片段：

```
…
while (!feof(fp1))// 文件未结束则继续循环，读、写数据
{
    fread(&stud,sizeof(struct student),1,fp1);
    fprintf(fp2,"%d,%s,%.1f\n",stud.num,stud.name,stud.score);
}
```

6.4.2　出错检测函数

格式：

int ferror(fp);

功能：检查文件在使用各种输入/输出函数进行读写时是否出错。当输入/输出函数对文件进行读写时出错，文件就会产生错误标志。

返回值：未出错函数返回值为 0，否则返回非 0，表示有错。

【例 6.15】 以读的方式打开例 6.10 中建立的学生数据文件 stud.dat，写入一个学生信息，利用 ferror 函数检查是否出错。

参考程序如下：

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
struct student
{
    int num;
    char name[20];
    char sex;
    int age;
    float score;
};
int main( )
{
    struct student stud;
    FILE *fp;
    long int where;
    if ((fp=fopen("stud.dat","rb"))==NULL)          // 以读方式打开二进制文件
    {
        printf("Cannot open file stud.dat!\n");
        exit(0);
    }
    stud.num=111;                                  // 保存学生信息
    strcpy(stud.name,"zhang");
    stud.sex='F';
    stud.age=20;
    stud.score=88;
    fwrite(&stud,sizeof(struct student),1,fp);     // 将学生信息写入文件中
    if (ferror(fp))                                // 检查写入文件是否出错
    {
        printf("File stud.dat write error!\n");
        exit(0);
    }
    fclose(fp);                                    // 关闭文件
    return 0;
}
```

此例中因为文件是以读的方式打开的，因此不可能写入数据，所以当调用 fwrite 函数写入数据时产生错误。

习　　题

一、选择题

1．下列叙述中正确的是（　　）。

A）C 语言中的文件是流式文件，因此只能顺序存取数据

B）打开一个已存在的文件并进行了写操作后，原有文件中的全部数据必定被覆盖

C）在一个程序中当对文件进行了写操作后，必须先关闭该文件然后再打开，才能读到第 1 个数据

D）当对文件的读（写）操作完成之后，必须将它关闭，否则可能导致数据丢失

2．如果没有执行则可能丢失数据的文件操作步骤是（　　）。

A）定义 FILE 类型指针　　B）打开文件

C）读写文件　　D）关闭文件

3．若执行 fopen 函数时发生错误，则函数的返回值是（　　）。

A）随机值　　B）1　　C）NULL　　D）EOF

4．要打开一个已经存在的非空文本文件 a.txt 用于修改数据，则正确的打开文件语句是（　　）。

A）fp=fopen("a.txt","r");　　B）fp=fopen("a.txt","w");

C）fp=fopen("a.txt","r+");　　D）fp=fopen("a.txt","w+");

5．假设变量已正确定义，则执行函数调用语句“fgets(ps,n,fp);”后从文件中读出的字符个数（　　）。

A）一定是 *n* 个　　B）最多为 *n* 个　　C）一定是 *n*−1 个　　D）最多为 *n*−1 个

6．读取二进制文件的函数调用形式为“fread(buffer,size,count,fp);”，其中 buffer 代表的是（　　）。

A）一个文件指针，指向待读取的文件

B）一个整型变量，代表待读取的数据的字节数

C）一个内存块的首地址，代表读入数据存放的地址

D）一个内存块的字节数

7．设有以下结构体类型数组的定义，且数组 mini 的 10 个元素都已赋值，若要将这些元素写到文件指针 fp 所指向的文件中，不正确的语句是（　　）。

```
struct abc
{
    int a;
    char b;
    float c[4];
}mini[10];
```

A）fwrite(mini, 10*sizeof(struct abc), 1L, fp);

B）fwrite(mini, 5*sizeof(struct abc), 2L, fp);

C）for (i=0; i<10; i++)

　　fwrite(mini, sizeof(struct abc), 1L, fp);

D）fwrite(mini, sizeof(struct abc), 10L, fp);

8．函数调用语句“fseek(fp,−10L,2);”的含义是将文件位置指针（　　）。

A）从文件头向后移动 10 个字节

B）从当前位置向后移动 10 个字节

C）从当前位置向前移动 10 个字节

D）从文件尾部向前移动 10 个字节

9．有以下程序：

```
#include <stdio.h>
int main( )
```

```
{
    FILE *f;
    f=fopen("a.txt","w");
    fprintf(f,"abc");
    fclose(f);
    return 0;
}
```

若文本文件 a.txt 中原有内容为 hello，则运行以上程序后，文件 a.txt 的内容为（　　）。

A）helloabc　　B）abclo　　C）abc　　D）abchello

二、填空题

1．在 C 语言中，根据文件的数据格式，把文件分为_____和_____两种类型。

2．若要通过变量 fp 来访问一个文件，则 fp 的定义形式为_____。

3．读文件时，为了检测文件指针是否已经到了文件尾部，需要使用的函数是_____（只写函数名）。

4．设有定义“FILE *fw;”，请将以下打开文件的语句补充完整，以便可以向文本文件 readme.txt 的最后续写内容。

```
fw=fopen("readme.txt",_____);
```

5．若有语句“fgets(str,20,fp);”，假设变量都已正确定义，则最多可以从 fp 指向的文件中读出_____个字符。

6．用 fread 函数或 fwrite 函数读写的文件必须是_____文件（填文件类型）。

三、程序填空题

1．若要以只读方式打开文本文件 a.txt，该文件位于 C:\Test 文件夹下，请填空。

```
FILE *fp;
fp=fopen("_____","_____");
```

2．若要把字符'A'写入到文件 b.txt 中，请填空。

```
FILE *fp;
fp=fopen("b.txt","w");
_____;
fclose(fp);
```

3．下面程序从键盘输入字符存放到文件中，用回车结束输入。存放字符的文件名也由键盘输入，请填空。

```
#include <stdio.h>
#include <stdlib.h>
int main( )
{
    FILE *fp;
    char ch,fname[20];
    printf("Input file name: ");
    scanf("%s",fname);
    if ((fp=fopen(_____,"w"))==NULL)
    {
        printf("Cannot open file.\n");
        exit(0);
    }
    printf("Input a string: \n");
    while (_____!='\n')
        fputc(ch, _____);
    fclose(fp);
    return 0;
```

```
}
```

4．下面的程序段用于从文本文件 score.txt 中读出长度最大为 20 的字符串，请填空。

```
FILE *fp;
_____;                          //定义字符数组 str
fp=fopen("score.txt","r");
if (fp==NULL)
{   printf("Cannot open score.txt!\n");
    exit(0);
}
fgets(str, _____,fp);
puts(str);
fclose(fp);
```

5．假设文件 data.dat 已经存在，数据存储格式为“10,20”，现要从该文件中依次读出两个整数赋给变量 a 和 b，请填空。

```
int a,b;
FILE *fp;
fp=fopen("data.dat","r");
fscanf(_____);
printf("%d %d\n",a,b);
fclose(fp);
```

四、编程题

1．从键盘输入一个字符串，以#结束，将其中小写字母全部转换为大写字母，然后输出到磁盘文件 letter.txt 中保存。

2．打开上题生成的文件，读入全部字符，输出到显示器上。

3．有两个文本文件 a.txt 和 b.txt，各存放了一个字符串。把这两个文件的信息合并后存放在一个新文件 c.txt 中。

4．在文件 r.dat 中存放了若干圆的半径（实数），从该文件中读取这些圆的半径，计算出相应圆的面积，并将结果存入 D 盘 area.dat 文件中。

假设文件 r.dat 的内容如图 6.3 所示。

```
1.0
2.0
3.0
4.0
5.0
6.0
```

图 6.3　r.dat 文件

5．建立一个磁盘数据文件 employee.dat，从键盘输入 3 个职工的数据存入该文件。每个职工的数据包括：职工工号、姓名、性别、年龄、家庭住址和工资。

6．从上题建立的 employee.dat 文件中读出职工信息，要求将职工姓名、工资信息提取出来另建一个简明的职工工资文件 emp2.dat。

第 7 章　实用程序设计技巧

程序设计课程是计算机科学教育的第一门专业课程，其目标是介绍如何理解程序语言，如何学习程序设计，如何为计算机领域中的进一步学习和工作做好准备。C 语言作为通用程序设计语言，由于它具有紧凑、高效、易于移植、模块化等特点，已被广泛地应用于系统程序设计、人工智能、文字处理、科学计算、计算机绘图等领域。尤其在编写系统软件或进行算法实现等领域，C 语言更以它的简洁、高效、易于与硬件交互等优点而被广泛采用。

本书讨论了基本程序设计的各方面问题，除了给出程序设计实例，帮助读者认识程序设计过程的实质，理解从问题到程序的思考过程以外，还将在这一章中进行与程序设计技巧有关问题的讨论。

本章首先讨论程序的模块化层次结构，引入软件工程的思想，介绍模块设计、分解和组装的原则和方法，然后再讨论大型 C 语言程序的设计风格和书写风格。

7.1　程序的模块化结构

程序设计也是一种工程性的工作，通过对任务进行分析和功能模块分解，将大任务分解为若干子任务，子任务分别进行设计之后，再进行组合，合并为功能强大而复杂的一个整体。可以把这些小功能模块看成“零件”，需要时就可以用这些“零件”组成“机器”，而不必每次都重新做起。这里就存在很多程序设计的技巧，可以加以总结和利用，从而大大地提高程序设计的效率，并且使编程者保持一个清晰的思路，而不被具体的细节捆住手脚。这里先引入软件工程的思想，分析如何进行模块分解。

7.1.1　软件工程的思想

系统设计分 4 方面的内容：体系结构设计、模块设计、数据结构与算法设计、用户界面设计。如果将软件系统比喻为人体，那么：

① 体系结构就如同人的骨架，它表明了人体的构成及各构件之间的关系。

② 模块就如同人的器官，具有特定的功能。模块和模块之间具有相对的功能独立性，同时也具有一定的联系。例如，手和脚的功能是相对独立的；嘴能与其他器官配合完成各种动作，如吃饭、说话等。

③ 数据结构与算法就如同人的血脉和神经，它让器官具有生命并能发挥功能。数据结构与算法分布在体系结构和模块中，它将协调系统的各个功能。

④ 用户界面就如同人的外表，最容易让人一见钟情。像人类追求心灵美和外表美那样，软件系统也追求（内在的）功能强大和（外表的）界面友好。

7.1.2　模块设计

在设计好软件的体系结构后，就已经在宏观上明确了各个模块应当具有什么功能，应该放在体系结构的哪个位置。一般从功能上划分模块，保持功能独立是模块化设计的基本原则。因为功能独立的模块可以降低开发、测试、维护等阶段的代价。但是功能独立并不意味着各模块之间保持绝对的孤立。一个系统要完成某项任务，需要各个模块相互配合才能实现，此时模块之间就要进行信息交流。

例如，手和脚是两个功能独立的模块。没有脚时，手照样能干活。没有手时，脚仍可以走路。但如果希望跑得快，那么迈左脚时一定要伸右臂甩左臂，迈右脚时则要伸左臂甩右臂。在设计一个模块

时不仅要考虑“这个模块就该提供什么样的功能”，还要考虑“这个模块应该怎样与其他模块交流信息”。评价模块设计优劣的 3 个特征因素是：信息隐藏、内聚与耦合和封闭–开放性。

以上形象化地描述了软件工程的思想和模块化的含义和分解特征，可以说模块化程序设计是软件工程思想的核心和主题。进行模块化设计时，通常将一个大型的程序自上向下地进行功能分解，分成若干个子模块，每个模块对应了一个功能，有自己的界面，有相关的操作，完成独立的功能。各个模块可以分别由不同的人员编写和调试，最后，将不同的模块组装成一个完整的程序。

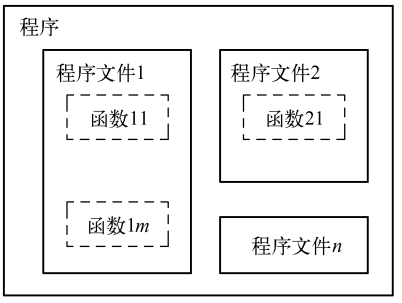

图 7.1　C 语言程序的模块化结构

在 C 语言中，用函数实现功能模块的定义，程序的功能可以通过函数之间的调用实现。一个完整工程项目的 C 语言程序可能由多个源程序文件组成，每一个文件中又可以包含多个函数。图 7.1 所示为一般的 C 语言程序的结构。关于函数的相关定义和调用知识我们已在第 3 章进行了学习和讨论，在此就不详细介绍了。

7.1.3　模块化的优点

把问题的解决方案细分为一系列的模块有许多好处。因为每个模块都有一个特殊的目的，所以就能撇开问题解决方案的其他部分而单独编写和测试该模块。独立的模块比完整的模块的解决方案要小，因此测试起来更加容易。同时，只要仔细测试过一个模块，不需要重新测试就可以直接将其用在新问题的解决方案中。例如，假设已经开发了一个模块来计算一组数字的最大值，只要编写并测试了此模块，就能在需要计算最大值的其他程序中使用它。复用性是大型软件系统开发中一个非常重要的问题，它可以节省大量的开发时间。事实上，在计算机系统中经常可以找到各种使用频繁的模块库（如标准 C 语言库），在 C 语言程序中也可以找到使用频繁的系统模块（如 scanf 输入函数和 printf 输出函数）。

模块的使用常常可以缩短程序的总长度。因为许多问题解决方案包含的步骤在程序中都是多次重复的，所以可以把这些重复的步骤都整合到一个函数中，每次需要它们时只需引用一条单独的语句即可。

如果一个项目被分割为几个模块，那么，同一个项目的开发工作就可以在若干程序员之间展开，每个独立的模块都可以单独地进行开发与测试。这样就能够加快开发进度，因为许多工作可以同时进行。

用于完成特定任务的模块支持抽象的概念，也就是说，模块内部包含了实现任务的具体细节，而对于使用该模块的程序或程序员而言，无须关注这些细节就可以引用该模块。在制订问题的解决方案时，使用的 I/O 图就是一个抽象的例子——指定输入信息和输出信息，但没有给出如何计算输出信息的细节。因此，可以把模块当成“黑匣子”，它们具有指定的输入，并具有指定的输出信息。在项目分解时可以先这样笼统地划分模块，这样，才可以通过更高级的抽象来解决问题，而无须一下子就关注所有细节。使用抽象，可以增强软件质量，同时也可以减少软件的开发时间。

总的来说，在问题解决方案中使用模块有以下优点：

- 模块可以独立于解决方案的其他部分进行单独的编写和测试，因此对于大型项目，各个模块的开发可以同步进行；
- 模块是解决方案的一小部分，因此单独测试起来更加容易；
- 经过仔细测试之后，不需要重新测试就可以将模块直接应用于新的问题解决方案中；
- 使用模块通常可以缩短程序的长度，使程序更具可读性；
- 模块的使用促使采用抽象的概念，从而允许程序员把细节“隐藏”于模块之中，这就使得用户

能够像使用系统库函数一样使用模块，而无须考虑具体的细节。

结构图或模块图显示了一个程序的模块结构。main 函数可以引用其他函数，而这些函数自身又可以引用其他函数。例如，图 7.2 中给出了一个语音信号分析系统的模块结构图例子。这里，main 主函数调用方差函数 std_dev、平方平均值函数 ave_power、绝对值平均值函数 ave_magn 和零交叉次数 crossings 函数，而方差函数 std_dev 中又调用了偏差函数 variance，在偏差函数 variance 中又调用了平均值函数 mean，后两个函数同时也被 main 函数直接调用使用。由此，可以看出这个系统的功能模块分解情况，系统主功能被分解为求平均值 mean、计算方差 std_dev、标准偏差 variance、平方平均值 ave_power、绝对值平均值 ave_magn 和零交叉次数 crossings 等 6 个功能模块。其中，模块 mean 和 variance 具有复用功能，又被其他功能模块调用。从图中可以看出整个系统模块的层次结构。有了这样的分析和模块层次结构，用户开发起系统来就会思路清晰，目的明确，模块的组装和功能调用也都有了明确的指导，若干模块同时开发，程序的开发效率大大提高。

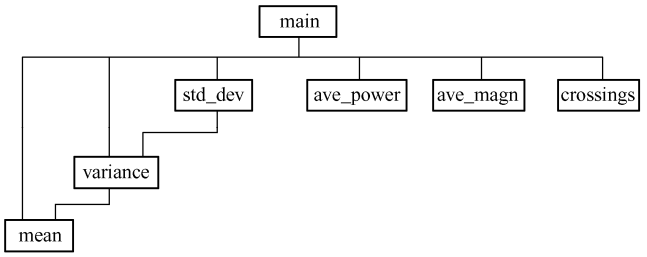

图 7.2　语音信号分析系统模块结构图

7.2　模块的组装

在用模块化方法开发程序时，一个完整工程项目的 C 语言程序通常会由多个源程序文件组成，每一个文件中又可以包含多个函数。模块的组装既涉及多个源文件的连接问题，也涉及实现具体模块的函数之间的连接调用关系。本节将讨论使用文件包含#include 命令实现多个源程序文件之间的连接，同时讨论模块之间的各种连接情况。

7.2.1　文件包含与头文件的使用

文件包含是指一个源文件可以将另一个源文件的全部内容包含进来，它是一个编译预处理命令。编译预处理命令是在程序编译之前进行的工作，不属于程序中的可执行语句，因此也不占用程序的运行时间。编译预处理命令都以“#”号开头，一般放在源程序首部，单独占源程序中的一行，行末不必加分号。C 语言提供的预处理指令主要有 3 种：宏、文件包含、条件编译。

文件包含命令的一般格式为：

```
#include <文件名>   或者   #include "文件名"
```

其中，文件名是指要被包含的源文件的文件名，可以是扩展名为.c 的 C 语言源程序文件，也可以是扩展名为.h 的库函数头文件。

在前面的例题中曾多次使用 include 命令包含系统库函数头文件，例如，使用 #include <stdio.h>，包含系统库函数头文件 stdio.h，这样就可以将此头文件中的全部内容包含到现有的文件中。stdio.h 头文件中大多都是系统定义好的关于输入/输出函数的函数声明语句，以及一些使用这些函数时需要用到的符号常量的定义，此外还有部分条件编译语句。使用了文件包含命令，就可以使用系统库函数功能，不必自己亲手编写许多模块功能。C 语言中类似于 stdio.h 这样的头文件还有很多，每一个头文件

均包含了某一类的系统功能，例如，string.h 头文件包含了字符串操作函数，math.h 包含了数学功能函数等。知道了这些，我们就可以充分利用已有的系统资源，从而提高程序的开发效率。

然而头文件的使用并不限于对系统资源的利用。在进行大型项目开发时，将程序功能分解为许多功能模块，模块通常都是用函数来编程实现的。在使用模块功能时，不需要将模块函数一一声明，通常会定义一些头文件，在头文件中进行相关函数的声明，或者声明一些许多模块都能用到的系统符号常量。这样在使用这些模块功能时，只需要包含已定义好的头文件，即可方便地使用这些模块功能。同时在头文件中定义符号常量，也便于整个系统所使用的符号常量的含义和功能统一，因为当不同的源程序文件包含了同一个头文件时，它们便拥有了相同的符号常量定义。

注意，头文件可以在 C 语言的源程序编辑环境中编写，也可以用记事本等文本编辑软件进行编写，但格式必须为文本文件。头文件的扩展名通常为.h。我们自己编写头文件时，文件保存的扩展名可以为.h，也可以是其他扩展名，但文件格式必须为文本文件。

用文件包含可以实现文件的拼接，即从磁盘中读取要包含的文件，然后把文件的内容插入到源程序中编译预处理命令行的位置上，使它成为源文件的一部分，然后再对合并的源文件进行编译。

说明：

① 使用文件包含时文件名两侧的< >与" "的区别是，< >表示直接到指定的标准包含文件目录中寻找包含文件；" "表示先在当前目录中寻找，如果找不到，再到标准包含文件目录中寻找。

② 使用" "时，头文件包含名可加路径。例如：

```
# include   "d:\\teach\\include\\compute.h"
```

③ 一个#include 命令只能指定一个被包含的文件，如果要包含 *n* 个文件，则要用 *n* 个#include 命令。

④ 如果 file2.c 中包含了 file3.c，而 file1.c 中需要包含使用 file2.c 和 file3.c 的内容，则 file1.c 直接包含 file2.c 即可使用 file3.c 中的内容。

【例 7.1】 分析以下程序的功能和结构。

参考程序如下：

```
// file.c 文件
#include "myhead.h"
int main( )
{
    int i,n=0;
    char str[100];
    gets(str);
    for (i=0;str[i]!='\0';i++)
    {
        if (isword(str[i])==TRUE)
            n++;
    }
    printf("There are %d words\n",n);
    getch( );
    return 0;
}
// myhead.h 的文件内容如下
#include <stdio.h>
#include "myfunc.c"
#define TRUE 1
#define FLASE 0
extern int isword(char c);

// myfunc.c 源文件内容如下
int isword(char c)
{
```

```
    if (c==' '||c=='\n'||c=='\t')
        return TRUE;
    else
        return FALSE;
}
```

以上实例中共有 3 个源文件。其中一个是 file.c，为主程序文件；一个是 myhead.h，是用户自定义的头文件；另一个是 myfunc.c，是实现模块功能的自定义函数文件，当然此文件中还可以包含多个自定义函数。

该实例的功能是统计一串字符中的单词个数，模块 isword 的功能是判断一个字符是否是单词结束字符，该函数代码单独作为一个文件存储在磁盘上，文件名为 myfunc.c。实例中，通过自定义的头文件 myhead.h 将模块功能函数包含在内，并进行有关的系统符号常量说明，然后在主程序文件 file.c 中通过包含头文件 myhead.h 并使用模块功能 isword 和系统符号常量 TRUE 和 FALSE 来判断单词并统计单词个数。

例 7.1 只是一个简单使用举例，在实际开发项目时，可以将具有相似功能的模块函数代码编写在同一个文件中，也可以将具体的模块代码编译成目标代码，以链接库的形式供主程序使用，同时定义相关的头文件进行相关函数的声明。这样系统就可以方便地使用由不同开发人员编写的各种模块功能了。

使用文件包含和头文件的好处如下：

① 可以充分利用系统资源，提高编写程序的效率；

② 可以实现多个源程序文件的拼接，将模块根据程序功能进行组装；

③ 将自定义函数的说明编写为头文件，然后包含此头文件，可使主程序简洁，模块结构明确；

④ 将不同功能的模块分别声明为不同的头文件，可以实现函数功能的自由组合。

7.2.2　模块间的连接

大型程序通常都由多个模块组成，除了文件包含的方法实现模块文件间的互相包含外，还有模块函数间通过参数或返回值等形式构成的连接。在进行模块组装时需要对模块间的连接进行充分的认识，以便在进行模块函数设计时，根据不同的情况选择不同的连接方式。

模块函数间的连接可以分为短暂连接和长久连接。所谓短暂连接是指只有在模块函数被调用时通过函数参数的传递或函数返回值和其他模块发生的连接关系，这是模块连接所采用最普遍的连接方式，也叫临时连接，当函数调用结束后连接关系就消失了。而长久连接是指通过全局变量或静态存储的变量和其他模块之间产生的连接关系。这种连接类似于人体的筋脉和血管等，使不同的模块之间产生相互的作用。这种连接是长期存在的，当函数一次调用结束后连接仍然存在，只有当整个程序运行全部结束时连接关系才取消。

模块间的短暂连接使得模块的功能比较独立，模块调用和组装非常灵活。通常模块间的短暂连接以 3 种形式存在：

① 普通参数：函数调用时主调函数以实际参数赋值给函数的形式参数，以完整模块功能。

② 返回值：函数调用结束后，返回指定的值给主调函数。

③ 指针参数：指针形式的参数使得函数调用时，函数通过指针直接访问主调函数中变量的内存单元，以此取得变量值或将函数处理结果放到指定的内存单元中。

【例 7.2】 银行存款年利率为 1.9%，编写程序计算并输出需要存款多少年才能翻一番，此时存款额是当初存入金额的多少倍？

分析：该程序中的函数模块需要返回两个值给主调函数，因为使用函数返回值只能返回一个值给主调函数，因此应考虑到使用函数返回值和指针参数共同传递数据的方法。

参考程序如下：

```
#include <stdio.h>
int function(double base,double *bei)
{
    int year=0;
    double temp=base;
    while (temp/base<2)
    {
        temp=temp*1.019;
        year=year+1;
    }
    *bei=temp/base;
    return year;
}
int main( )
{
    double base,x;
    int age;
    printf("Input base:");
    scanf("%lf",&base);
    age=function(base,&x);
    printf("after %d years, it is %.2f\n",age,x);
    return 0;
}
```

例 7.2 中函数模块 function 和主程序 main 之间使用了 3 种不同形式的数据连接；参数 base 使用的是普通参数传递，值的传递方向是将主调函数 main 中的变量值 base 传递给函数的形参 base；函数返回值返回的是年数，值的传递方向是由函数 function 传递给 main 函数；参数 bei 使用的是指针参数，为了将第 2 个返回值返回给 main 函数，在例 7.2 中发生函数调用时，参数 bei 指针被赋值为&x，所以指针 bei 指向 main 中的变量 x 的内存单元，所以在函数中间接访问了 x 的内存单元，将值 temp/base 间接放入了 x 的内存空间。因为是地址访问，所以参数 bei 的数据传递是双向的。

以上模块间的连接均使用了短暂连接，也就是当函数模块 function 被调用时才产生数据连接，当函数模块调用结束后，数据连接就断开不存在了。在实际使用时应根据情况选择数据连接的形式，如果仅仅是需要从主调函数获得参数来完成函数功能，则选择普通参数；如果函数模块只需返回一个值给调用它的模块，则选择函数返回值；如果从主调模块中获得某个变量值完成函数功能，同时还需改变此变量值，则可将这个参数定义为指针参数，或者如果模块需要返回多个值给主调模块，则需增加指针类型的参数，用以将一个以上的结果值返回给主调模块。

模块间的长久连接使得模块间衔接紧密，模块间的耦合加强，使得模块的独立性下降，各模块间通过全局变量等形式的连接产生了相互的影响。在某些情况下可以使用此种形式的连接，例如，模块对统一的系统变量产生作用，使系统变量的值发生改变，而这种改变需要保留到下一个程序运行时刻等。一般情况下不采用长久连接形式来完成模块间的连接和组装。模块间的长久连接有如下两种表现形式。

① 全局变量

全局变量又称为外部变量，是定义在函数外部的变量，可以被若干个函数模块所访问。每一个程序模块访问全局变量改变其值后，全局变量的值就发生了永久的改变，即函数调用结束后这种值的改变仍然生效。全局变量又分为文件内部的全局变量和文件间的全局变量。文件内部的全局变量的作用域是从定义它的位置开始直到源文件结束，即在此段之间的函数都可访问该全局变量。文件内部的全局变量使得一个文件内部的函数之间可以相互影响和作用。文件间的全局变量是指将全局变量声明为 extern，以供多个源程序文件访问，这样就将全局变量的作用域范围扩大了，扩大到了多个文件内部，即在多个源程序文件通过全局变量实现了连接。实际使用时需要根据系统功能要求有选择的使用全局

变量。使用全局变量时一定要慎重，因为任何模块对全局变量的修改都是长久的，这样有可能会带来副作用，使用不同模块访问全局变量时，对值的大小产生歧义。

② static 静态存储类

函数内部用 static 声明的静态变量是存储在系统的静态存储区的，其生存期较长，不随函数的调用结束而释放，而是在整个程序运行期间一直都保持有效。这样，该函数的多次被调用通过静态变量和主调函数之间，或多个不同的主调函数之间，产生了长期的作用关系。这种改变虽然是反映在函数内部的静态变量上的，但也是长期存在的，因此也属于模块间的长久连接。在编程时要根据情况有选择地使用。在特殊的情况下巧妙地利用函数的静态变量可以实现特殊的功能，但大多数情况下应避免使用静态变量使函数间产生长久连接关系。

【例 7.3】 观察分析以下程序中模块间的连接使用了哪种形式。

```
float max=0,min=100;                          // 定义全局变量
float average(int n)                          // 定义函数 average
{
    int i;
    float s,aver1,sum=0;
    for (i=1;i<=n;i++)
    {
        printf("请输入一个学生的成绩：");
        scanf("%f",&s);
        if (s>max) max=s;
        if (s<min) min=s;
        sum=sum+s;
    }
    aver1=sum/n;
    return(aver1);
}
int main( )
{
    int n;
    float aver2;
    printf("请输入班级人数：");
    scanf("%d",&n);
    aver2=average(n);                         // 函数调用
    printf("平均分为%6.2f,最高分为%6.2f,最低分为%6.2f\n",aver2,max,min);
    return 0;
}
```

例 7.3 中函数模块 average 和主程序 main 之间使用全局变量 max 和 min 构成了长久的数据连接。函数 average 被调用时，全局变量 max 和 min 的值在函数模块中被修改了，当模块 average 调用结束后，全局变量 max 和 min 被修改的值仍然有效，可以在主函数 main 中继续使用。

全局变量数据连接在使用时需谨慎。一是，当函数内部有与全局变量同名的局部变量时，全局变量会被屏蔽，也就是系统默认使用的是函数内部的局部变量，因此应注意函数内部变量的命名与全局变量的命名相同产生的冲突。二是，不同模块都可以访问全局变量修改全局变量的值，容易产生不必要的副作用。

【例 7.4】 观察分析以下程序中全局变量副作用的结果。

```
#include <stdio.h>
int i;                                        // i 为全局变量
int main( )
{
    void prt( );
    for (i=0;i<5;i++)
```

```
        prt( );
    return 0;
}
void prt( )
{
    for (i=0;i<5;i++)
        printf("%c",'*');
    printf("\n");
}
```

程序本意是输出 5 行星号，由于 i 被定义为全局变量，当 prt 函数被调用时 i 被改为 5，主函数 main 中的 for 循环实际上只执行了一次，因此实际程序运行只输出了一行星号。

7.2.3　标识符的一致性

在由许多分别编译的单元所组成的程序中，需要考虑的问题是，如何保证在各个单元中使用的标识符的每一个实例都能正确地与其所指定的程序实体相联系。使它们在各编译单元中的名字和类型的定义与使用时严格一致，这是模块连接成功与否的关键。

程序的一致性尽管原则上可以由一些高级的连接程序（也称装载程序）来保证，但是 C 语言鼓励程序员进行显式连接，以防不测。

C 语言系统进行编译时，要对统一编译单元（同一源文件）内使用的变量及函数的类型进行一致性检查。当程序由多个文件组成时，要对不同模块内定义和使用的变量及函数的一致性进行检查，以便实现类型的安全连接。一个模块中的程序实体的标识符的连接属性用于确定该实体是否可被其他模块引用。每个标识符要么具有连接性，要么无连接性。

连接属性与标识符的作用域密切相关，并且由声明的布局与格式决定。对于动态变量是无连接性的，具有连接性的变量是静态变量和全局变量。全局变量的作用域范围较大，属于公有变量，可以被许多函数访问使用，但注意当函数内部有与全局变量同名的局部变量时，系统默认使用的是函数内部的局部变量，因此，应注意函数内部变量的命名与全局变量的一致性的冲突。对于静态变量，其作用域范围仅限于本函数内部，属于私有变量，但其生存期较长，在整个程序运行期间均有效。这一点应加以注意。

在实际开发系统时，应注意标识符的一致性问题，在进行模块设计初期就要考虑到标识符间的连接属性和标识符的一致性问题。当某个标识符在各模块间具有较强的连接作用时，该标识符的意义和类型一定要一致，而且要避免函数内部变量的命名与其发生同名冲突。当某些标识符需要被某个特定的模块单独且长期地使用时，应该把它定义为私有性质的变量，即声明为函数内部的 static 类型变量。

7.2.4　条件编译

在一般情况下，源程序中所有的内容都参加编译，但是有时希望其中一部分内容只在满足一定条件下进行编译，也就是对一部分内容指定编译条件，这就是条件编译。

条件编译可以根据实际的系统特征或运行要求，进行有选择的编译，从而使系统适应性强，软件功能更为灵活。

条件编译命令包括以下几种形式。

（1）形式 1

```
#ifdef 标识符
    程序段 1
#else
    程序段 2
#endif
```

其作用是，当标识符已经被定义过（一般用#define 命令定义），则对程序段 1 进行编译，否则对程序段 2 进行编译。其中，#else 可以没有，即

```
#ifdef 标识符
    程序段 1
#endif
```

源程序段部分可以是任何预处理命令、C 语句或其他任何语法成分。

（2）形式 2

```
#ifndef 标识符
    程序段 1
#else
    程序段 2
#endif
```

其作用是，当标识符未被定义则对程序段 1 进行编译，否则对程序段 2 进行编译。其中，#else 可以没有，它的作用恰好与第一种形式相反。

（3）形式 3

```
#if  表达式
    程序段 1
#else
    程序段 2
#endif
```

其作用是，当指定的表达式值为真时编译程序段 1，否则编译程序段 2。

【例 7.5】 求一组整数的最大值、最小值及平均值。通过条件编译使本程序在调试和正式运行时具有不同特征和功能。

在设计程序时，应考虑到调试程序和正式运行程序的不同。在调试程序时，数据输入要少、要简单，以便加快程序调试过程。同时输出信息要多，有利于分析程序。正式运行程序时，输入数据要真实，不输出某些中间结果。

参考程序如下：

```
#include <stdio.h>
#define N 10
#define TEST 1
int main( )
{
    int i,max,min,a[N];
    float avg;

#if TEST
    for (i=0;i<N;i++)        // 调试时通过赋值得数组 a 的各元素值
        a[i]=i+5;
#else
    // 正式运行通过输入，得数组 a 的各元素值
    printf("Please Input array a[0]~a[%d]:\n",N-1);
    for (i=0;i<N;i++)
        scanf("%d",&a[i]);
#endif
    max=a[0];   min=a[0];   avg=a[0];
    for (i=0;i<N;i++)
    {
        if (max<a[i])   max=a[i];
        if (min>a[i])   min=a[i];
        avg=avg+a[i];
    }
```

```
#if TEST
    printf("sum=%f\n",avg);      // 调试时查看累加和，运行时不再查看
#endif
    printf("max=%d, min=%d, avg=%f\n",max,min,avg/N);

    return 0;
}
```

定义符号常量 test，如果是调试程序，将 test 置 1，则通过条件编译使数组各元素的值通过赋值形式快速获得，同时输出程序的中间结果，即为了求平均值首先得到的累加和。而正式运行程序时则编译另外部分的代码，使数组各元素的值通过输入函数，让用户从键盘输入，同时也省略了部分中间结果（即累加和）的输出。

7.3　模块设计风格简述

模块设计风格是指实现模块功能代码时经常采用的一些设计风格。在编程时应注意养成良好的编程习惯和编程风格，良好的模块设计风格可以使程序便于阅读、理解和调试，从而提高编程效率。在大型程序开发时，注重编程风格尤为重要。大型程序通常是由不同的开发人员编写的，不同编程人员编写的程序代码必须能够相互理解，相互通融，表达一致，这样必然要有一个统一、良好的模块设计风格。

模块的设计风格包含很多方面，在此主要从模块的数据风格、标识符风格、算法风格、输入/输出风格、书写风格等几方面加以总结和讨论。

7.3.1　数据风格

编程时所使用的数据类型和数据结构要清晰、明确，同时尽可能地用接近现实意义中的类型来表达。例如，求圆的面积时，定义的圆半径变量和圆面积变量根据现实意义应定义为实数类型。

指针类型是 C 语言特有的数据类型，它使得程序可以通过指针直接访问某些内存单元。指针的使用一方面增强了程序的功能，同时也带来了一定的副作用。如果对指针的含义弄不清楚，或二级指针等复杂使用，容易出现错误的内存单元访问情形，导致程序运行结果错误，甚至对系统造成破坏，因此应有限制地使用指针。

使用数组形式来操作一组数据时，注意数组的长度与实际数据长度的关系，尤其是对于字符数组，应考虑到各种情况的数组长度包容关系，同时又尽可能地节省内存空间。

在用结构体和共用体类型来表达复杂结构的数据类型时，要注意类型定义的一致性，通常将数据类型的定义放在头文件中，或放在函数外部定义，这样定义的类型具有公有性质，可以被众多模块访问使用。

对于构造数据类型或对于具体开发来说有特殊意义的基本数据类型，可以使用 typedef 命令重新定义类型名，以便与现实意义相统一，方便理解。

7.3.2　标识符风格

1．见名知意

在进行变量命名或函数命名时，应注意做到见名知意，即选择有含义的英文单词（或其他缩写）作为标识符，如 count，name，day，class，city，area 等。同时可以采用较长的描述性名字来命名变量或函数等，如驼峰式命名法或加下画线命名法。

例如：

PrintEmployeePaychecks

Print_Employee_Paychecks

2．附加类型关键字

变量标识符可在变量名后加一条下画线，后面附上该变量的类型（或类型缩写），这样容易理解变量类型和含义，使程序容易阅读和调试。例如，变量 day_int，sum_float 等。

3．常用函数命名规则

在对函数命名时，可以采用主体模块的缩写以表示该模块的功能，如“age_day;”、“age_year;”、“age_month;”等，或采用动宾结构来描述函数标识符，如“ResetCounter;”、“ReleaseLock;”等。

4．使用符号常量

符号常量是一种无参宏，用#define 命令定义。有时为了提高程序的可读性、增强程序的可修改性，可在程序开始之处用符号常量的形式来表达某种常量信息。如圆周率π值、数组长度的定义 N 等，均可在主程序开始处定义为符号常量。同时对于多个模块都要访问的具有现实意义的常量，也应定义为符号常量，供多个模块访问，这样含义明确，易于理解。

7.3.3　算法风格

① 算法要简洁明了，尽量少使用技巧。

例如，两个变量互换值的算法可以有两种算法实现。

- 算法 1：a=a+b; b=a–b; a=a–b;
- 算法 2：定义一个临时变量 temp，temp=a; a=b; b=temp;

比较上述两种算法，算法 2 简洁明了，易于阅读和理解。

② 尽量避免使用多重循环嵌套或条件嵌套结构。

③ 在条件或循环结构中尽量避免采用“非”条件测试。

④ 尽量避免复杂条件测试。

⑤ 语句和表达式要清晰、易读。例如，表达式 x=(y=2)+(z=5); 写成下面的形式会使算法更清晰且容易理解：

```
y=2; z=5; x=y+z;
```

⑥ 充分利用系统库函数功能。

⑦ 要注意浮点运算误差，在编程时对于实数类型的数据要避免进行精确相等的比较。例如，10.0*0.1 一般与 1.0 不相等。又如，实数与 0 的比较通常要近似表达为 fabs(a)<=1e–6 等。

7.3.4　输入/输出风格

输入/输出风格决定了用户可以接受程序的程度。好的输入/输出风格能使用户工作在轻松的环境之中，提高他们的工作效率。在程序设计时，考虑输入/输出风格的具体做法如下。

① 提高输入操作的坚固性，以适当的方式对输入数据进行检验，以确认每个输入数据的有效性。对无效数据，也能给出必要的提示，而不导致死机或输出错误结果。

② 输入格式简单、单一、统一，容易核对。

③ 输入格式与用户水平相对应。

④ 输入时能给用户以提示，指明可使用的选择和边界值。

⑤ 输出格式应满足用户要求，符合使用意图。

⑥ 对输出操作有必要的提示。

⑦ 简化用户操作，减少用户出错处理。

7.3.5　书写风格

程序代码的书写风格的核心目标是提高程序的可读性。一般来说包括以下几个主要方面：

1．使用足够的注释

为了帮助阅读理解程序，应当使用足够的注释。特别要注意在下列地方使用注释：

- 一个文件的文件名
- 程序或函数的功能
- 变量的用途
- 特殊数据结构的特点和实现方法
- 特殊算法技巧
- 任何容易误解或别人不容易看懂的地方

2．缩进格式

对于层次不同的控制结构，分别缩进几格书写，可以使程序结构层次清晰，容易观察程序的控制结构和进行算法分析。例如，如下程序段的缩进格式：

```
#define LEN sizeof(struct student)
struct student *creat(void)
{
    struct student *head,*p1,*p2;
    float s;
    n=0;
    head=NULL;                                  // 初始化链表为空
    p1=p2=(struct student *)malloc(LEN);        // 开辟第一个新单元
    scanf("%ld%f",&p1->num,&s);
    p1->score=s;                                // 读入第一个结点的数据
    while (p1->num!=0)
    {
        n=n+1;
        if (n==1)
            head=p1;
        else
            p2->next=p1;
        p2=p1;
        p1=(struct student *)malloc(LEN);       // 开辟新结点
        scanf("%ld%f",&p1->num,&p1->score);     // 读入新结点的数据
    }
    p2->next=NULL;                              // 链表建立结束
    return(head);
}
```

3．括号风格

（1）花括号

使用缩进书写格式，同时选择统一的语句花括号将有关联的多条语句包括为一个整体，最能体现具体的模块功能，如分支模块、循环模块等，可以突出结构的层次关系。

（2）圆括号

使用冗余的圆括号使表达式易读。例如，把表达式“c=b*=a+2;”，改写为“c=(b*=(a+2));”。

7.4　应用程序设计实例

【例 7.6】 从 *n* 个不同价值、不同质量的物品中选取一部分，在不超过限定的总质量的前提下，使该部分的价值最大。这里假定的总质量不超过 *n* 个物品的总质量之和，且没有一样物品的质量超过限定的总质量。

分析：这个问题是求最佳解的典型例子。为找到最佳解，需生成所有可能的解。在生成这些解的同时，保留一个指定意义下的当前最佳解，当发现一个更好的解时，就把这个解改为当前的最佳解，并保留之。

现给出一组 *n* 个物品中找出满足约束条件的最佳解的通例。为便于构造算法，采用递归方法。构成可接受解的所有选择是通过依次考察组中的各个物品的结果。对每个物品的考察均有两种可能，或所考察的物品被包括在当前选择中，或所考察的物品不被包括在当前选择中。递归函数是描述指定物品被包括或不被包括在当前选择中的计算过程，只要指定物品被包括后质量满足约束条件，该物品被包括就是应该被考虑的；仅当一个物品如果不被包括也可能达到比当前最佳解所达到的总价值大时，为满足质量的限制，不把该物品包含在当前选择中也是应该被考虑的。为此，递归函数设有 3 个参数：指定的物品、当前选择已达到的总质量和可能达到的总价值。下面的递归算法就是考察某个物品在当前选择中是否被包括的计算过程描述。

算法描述——物品 i 在当前选择中被包括与否的递归算法：

```
try(物品 i,当前选择已达到的总质量 tw,可能达到的总价值 tv)
{
    if (包含物品 i 是可接受的)              // 考察当前选择包含物品 i 的合理性
    {
        将物品 i 包括在当前解中;
        if (i<n−1)
            try(i+1,tw+物品 i 的质量,tv);
        else
            调整当前最佳解;
        将物品 i 从当前解中消去;
    }

    if (i<n−1)                              // 考察当前选择不包含物品 i 的合理性
        try(i+1,tw,tv−物品 i 的价值);
    else
        调整当前最佳解;
}
```

对当前选择而言，“包含物品 i 是可接受的”准则是它被包括后，有可能达到的总价值也不小于当前最佳解所达到的价值，因为如果小于，继续考察下去也不会产生更好的解。

参考程序如下：

```
#include <stdio.h>
#define N 100
double      limw,       // 物品的约束质量
            totv,       // 全部物品的总价值
            maxv;       // 解的总价值
int         opts[N],    // 当前最佳选择
            cs[N],      // 当前选择
            n,          // 物品数
            k;          // 工作变量
struct goods
{
```

```
    double weight;    // 物品的质量
    double value;     // 物品价值
}a[N];                // 用于存放一组物品
void tryy(int i,double tw,double tv)
{
    // 考察当前选择物品 i 的合理性
    if (tw+a[i].weight<=limw)                    // 包含物品 i 是可接受的
    {
        cs[i]=1;                                 // 将物品 i 包括在当前解中
        if (i<n−1)
            tryy(i+1,tw+a[i].weight,tv);
        else if (tv>maxv)                        // 调整当前最佳解
        {
            for (k=0;k<=i;k++)
                opts[k]=cs[k];
            maxv=tv;
        }
        cs[i]=0;                                 // 将物品 i 从当前解中消去
    }
    // 考察当前选择不包含物品 i 的合理性
    if (tv−a[i].value>maxv)                      // 不包含物品 i 是可接受的
        if (i<n−1)
            tryy(i+1,tw,tv−a[i].value);
        else                                     // 调整当前最佳解
        {
            for (k=0;k<=i;k++)
                opts[k]=cs[k];
            maxv=tv−a[i].value;
        }
}
int main( )
{
    printf("Input number of mails:\n");
    scanf("%d",&n);
    printf("Input limit of weight:\n");
    scanf("%lf",&limw);
    printf("Input weight and value of mails:\n");
    for (k=0;k<n;k++)
        scanf("%lf%lf",&a[k].weight,&a[k].value);
    for (totv=0.0,k=0;k<n;k++)
        totv+=a[k].value;
    maxv=0;
    for (k=0;k<n;k++)
        opts[k]=cs[k]=0;
    tryy(0,0,totv);
    for (k=0;k<n;k++)
        if (opts[k])
            printf("%4d",k+1);
    printf("\nTotal value=%lf\n",maxv);
    return 0;
}
```

【例 7.7】 聪明的小老鼠。有 37 只小老鼠围成一圈，编上号码（1～37），一只大猫想要吃掉这些老鼠。大猫从 1 开始数，吃掉编号为 5 的老鼠，接下来从后面的那只老鼠为 1 开始数，数到 5 仍然吃掉，……，最后只剩下一只小老鼠。问编号为几的老鼠活了下来？

方法一：用循环算法完成。

参考程序如下：

```
#include <stdio.h>
#define max 37
int main( )
{
    int a[37],i,j=37,m=0,k,q;
    for (i=0;i<37;i++)              // 初始化部分
        a[i]=i+1;                   // 将数组元素依次赋值 a[0]=1，a[1]=2…，相当于为老鼠编号
    i=0;

    for (k=1;k<=36 ;k++)            // 37 个老鼠要吃掉 36 个，循环 36 次
    {
        // 数数部分
        for (m=1;m<=4;m++)  // 数数
        {
            i++;
            if (i>=j)               // 数到头了又从第一个开始数，循环数数
                i=i-j;
        }

        // 移位部分，活着的要顶替死的位置
        q=i;
        while (q<j)                 // 吃掉一个后，后面的依次往前移一位
        {
            a[q]=a[q+1];
            q++;
        }
        j--;                        // 吃一个则个数少 1
    }
    printf("活着的老鼠是第%d 位.",a[0]);
    return 0;
}
```

方法二：用链表结构完成。

```
#include <stdio.h>
#include <malloc.h>
typedef struct node                 // 定义结点类型
{
    int data;
    struct node *next;
    int mark;                       // 死亡标志，0 表示未被吃掉，1 表示已被吃掉
}NODE,*LINK;
LINK great_link( )                  // 建立一个包含 37 个结点（老鼠）的循环链表
{
    LINK k;
    NODE *p,*start;
    int i;
    k=(NODE*)malloc(sizeof(NODE));
    start=k;
    for (i=0;i<37;i++)              // 建立包含 37 个结点的循环链表
    {
        p=(NODE*)malloc(sizeof(NODE));
        k->next=p;
        p->next=start;
        k=p;
    }
    return k;                       // 返回链表头结点
}
LINK install(LINK L)                // 链表初始化，给老鼠编号
```

```
{
    int i;
    NODE *p;
    p=L;
    for (i=1;i<=37;i++)                    // 用 1 到 37 编号
    {
        p->data=i;
        p->mark=0;
        p=p->next;
    }
    return L;
}
LINK survive(LINK L)                       // 查找幸存的老鼠
{
    NODE *p;
    int count=0,dead=0;                    // count 用来数数；dead 用来统计被吃掉的老鼠数
    p=L;
    while (1)
    {
        if (p->mark==0)                    // 死亡标志为 0 表示没有被吃掉
            count++;
        if (count==5)
        {
            p->mark=1;                     // 将该老鼠的死亡标志置为 1
            count=0;                       // 重新开始计数
            dead++;                        // 死亡数量 dead 累计
        }
        p=p->next;
        if (dead==36)                      // 已经死掉 36 只老鼠就终止循环
            break;
    }
    return L;
}
void print(LINK L)                         // 输出没有被吃掉的老鼠
{
    NODE *p;
    p=L;
    while (1)
    {
        if (p->mark==0)
        {
            printf("number %d isnot dead.\n",p->data);
            break;
        }
        p=p->next;
    }
}
int main( )
{
    LINK L,k,T;
    L=great_link( );
    k=install(L);
    T=survive(L);
    print(T);
    return 0;
}
```

【例 7.8】 计算氨基酸化学式的分子质量。已知各原子的质量如下：H 原子 1.00794，C 原子

12.011，N 原子 14.00674，O 原子 15.9994，S 原子 32.0666。

分析：程序输入的是化学式，可把它作为一个字符串存在一个数组中。逐个分析字符确定各元素，并确定每个元素原子的数量。因为这种比较需要几个步骤，将这种分析计算在一个单独的函数中实现。在 main 函数中，将当原子的质量乘以原子数，然后将各个值加在一起得到总质量。

参考程序如下：

```
#include <stdio.h>
#include <ctype.h>
#define NEWLINE '\n'
double atomic_wt(int atom);                                    // 函数声明
int main(void)
{
    int k=0,formula[50],n,current=0,done=0,d1,d2;
    double error=0,weight,total=0;
    printf("Input chemical formula for amino acid:\n");
    while ((formula[k]=getchar( ))!=NEWLINE)                   // 从键盘读入化学分子式
        k++;
    n=k;
    while (current<=n-1&&done==0)                              // 辨别化学元素，累加质量
    {
        if (isalpha(formula[current]))                         // 当前字符为字母或数字
        {
            formula[current]=toupper(formula[current]);        // 把字母修改为大写
            weight=atomic_wt(formula[current]);                // 计算该原子的质量
            if (weight==0)                                     // 非法字符，结束计算
                done=1;
            else
            {
                if (current<n-1)                               // 若字符串未结束
                    d1=isdigit(formula[current+1]);            // 下一位为数字字符
                else
                    d1=0;
                if (d1 && current<(n-2))                       // 检查是否有第 2 位数字
                    d2=isdigit(formula[current+2]);            // 原子个数包含第 2 位数
                else
                    d2=0;
                if (d1 && d2)                                  // 原子个数为两位数
                {
                    weight*=((formula[current+1]-'0')*10+(formula[current+2]-'0'));
                    current+=3;
                }
                else if (d1)                                   // 原子个数为 1 位数
                {
                    weight*=(formula[current+1]-'0');
                    current+=2;
                }
                else                                           // 原子个数为 1 个
                    current++;
            }
            total+=weight;                                     // 累加质量
        }
        else                                                   // 当前字符为非字母或数字
            done=1;                                            // 结束计算
    }
    printf("Formual:\n");                                      // 打印化学式和质量
    for (k=0;k<=n-1;k++)
        putchar(formula[k]);
```

```
    printf("\n");
    if (done==0)                                          // 化学式无误则输出质量
        printf("Molecular Weight:%f\n",total);
    else                                                  // 化学式错误输出提示信息
        printf("Error in formula. \n");
    return 0;
}

double atomic_wt(int atom)                                // 计算一个原子的质量
{
    int k=0,element[5]={'H','C','N','O','S'};            // 假设包含这 5 种原子
    double m_wt[5]={1.00794,12.011,14.00674,15.9994,32.066},weight;
    while (k<=4 && element[k]!=atom)
        k++;
    if (k<=4)
        weight=m_wt[k];
    else
        weight=0;
    return weight;
}
```

【例 7.9】 一个简单的图书借阅程序。图书信息包含以下数据项：图书编号、图书名、出版社、出版时间、是否已被借阅。

要求：

① 根据以上信息定义图书的结构体类型 book。

② 假定该图书馆有图书 N 本，定义该结构体类型数组。程序运行时，先从键盘上输入图书信息，建立该图书信息库。

③ 由用户从键盘上输入所借阅的“图书编号”或“图书名”，程序根据输入信息，查找有无该图书，如果没有则显示“没有该图书”。如果有该书，则查看该书是否已被借阅（最后一个成员值），如果已借阅则反馈信息为“该书已借出，不能借阅”；如果没被借阅，则将该书借出（借阅标志变为'Y'）并显示“借阅成功!”。

④ 还书功能。给出图书名称，查找并修改将该书的“借阅”成员，将'Y'改为'N'。

“输入图书信息”、“图书借阅”和“还书”用函数实现，函数名分别为 input、borrow 和 back。input 函数的参数有一个结构体类型的指针（或结构体类型的数组）；borrow 函数的参数为两个，一个为结构体指针，一个为图书书名。back 函数的参数为两个，一个是结构体数组（或指针），一个是图书的书名。

参考程序如下：

```
#include <stdio.h>
#include <string.h>
#define N 20                    // 图书的数量

struct book
{
    int num;                    // 图书编号
    char name[50];              // 书名
    char publish[40];           // 出版社
    struct                      // 出版日期
    {
        int year;
        int month;
    }date;
    char borrow;                // 是否被借阅标志
```

```
};

void input(struct book *b);                          // 图书信息录入函数
void output(struct book *b);                         // 显示图书信息函数
void borrow(struct book *b, char name[ ]);           // 图书借阅函数
void back(struct book *b, char *name);               // 还书函数

int main( )
{
    struct book books[N];                            // 定义 N 本图书的结构体数组
    char name[50];                                   // 存放图书名
    input(books);                                    // 输入图书信息
    output(books);                                   // 显示图书信息
    printf("Now you can input the book name you are going to borrow:\n");
    scanf("%s",name);                                // 输入要借阅的图书名
    borrow(books,name);                              // 借阅图书
    output(books);                                   // 显示图书信息
    printf("Now you can input the book name you are going to return:\n");
    scanf("%s",name);                                // 输入要还的图书名
    back(books,name);                                // 还书
    output(books);                                   // 显示图书信息
    return 0;
}

void input(struct book *b)                           // 或为 void input(struct book b[ ])
{
    int i;
    printf("Input book\'s num,name,publish,date,borrow:\n");
    printf("Be caution: you should input the <Space> to seperate every item!\n");
    for (i=0;i<N;i++,b++)
    {
        printf("number %d:",i+1);
        scanf("%d",&b->num);
        scanf("%s %s",b->name,b->publish);
        scanf("%d%d %c",&b->date.year,&b->date.month,&b->borrow);
    }
}

void output(struct book *b)                          // 显示库中的图书信息
{
    int i;
    printf("\n Output the book\'s information:\n");
    printf("-----------------------------------\n");
    printf(" num      name      publish      publishdate      borrow:\n\n");
    for (i=0;i<N;i++,b++)
        printf("%-10d%-16s%-12s%6d,%d%9c\n",
                b->num,b->name,b->publish,b->date.year,b->date.month,b->borrow);
    printf("-----------------------------------------------------\n\n");
}
void borrow(struct book *b, char name[ ])            // 借阅图书
{
    int i,flag=0;
    for (i=0;i<N;i++,b++)                            // 查找图书
    {
        if (strcmp(b->name,name)==0)                 // 找到要借的书
        {
            flag=1;                                  // 将找到标志置为 1
            if (b->borrow=='N')                      // 如果该书没有被借阅，则修改借阅标志
```

```
                {
                    b->borrow='Y';
                    printf("Success!\n");
                }
                else                                    // 该书已被借走，输出提示信息
                    printf("The book has been borrowed.You can\'t borrow it.\n");
            }
        }
        if (flag==0)                                    // 库中没有要借的书
            printf("We can\'t find this book.\n");
}
void back(struct book *b, char *name)                   // 还书
{
    int i,flag=0;
    for (i=0;i<N;i++,b++)
    {
        if (strcmp(b->name,name)==0)                    // 找到该书
        {
            flag=1;                                     // 将找到标志置为 1
            b->borrow='N';                              // 修改借阅标志为未借阅状态
            printf("The book has been returned.\n");
        }
    }
    if (flag==0)                                        // 如果查找标志为 0，则输出没有找到的提示
        printf("Can\'t find this book! The book doesn\'t belong to our library.\n");
}
```

习　　题

一、选择题

1．C 语言中使用（　　）来实现模块的定义。

A）语句　　B）函数　　C）过程　　D）文件

2．评价模块设计优劣的特征因素不包括（　　）。

A）功能独立　　B）内聚与耦合　　C）信息隐藏　　D）封闭-开放性

3．下列关于模块化程序开发的说法不正确的是（　　）。

A）每个模块都有一个特定的功能

B）可以缩短程序的总长度

C）可以由若干程序员独立开发各模块

D）由于是不同的程序员开发的，因此测试变得更加复杂

4．关于编译预处理命令的说法不正确的是（　　）。

A）C 语言的预处理命令主要有：宏、文件包含和条件编译

B）预处理命令不占用程序的运行时间

C）一行可以写多个预处理命令

D）预处理命令的末尾不加分号

5．不能实现模块间短暂连接的是（　　）。

A）普通参数　　B）返回值　　C）指针参数　　D）局部变量

6．关于模块间的长久连接说法不正确的是（　　）。

A）使得模块间的衔接更加紧密

B）模块的独立性增加了

C）使用全局变量可以实现模块间的长久连接

D）使用 static 局部变量可以实现模块间的长久连接

7．下列程序段的作用是（　　）。

```
#ifdef DEBUG
    printf("x=%d\n",x);
#else
    printf("y=%d\n",y);
#endif
```

A）如果定义了标识符 DEBUG，将输出 x 的值

B）如果定义了标识符 DEBUG，将输出 y 的值

C）如果标识符 DEBUG 的值为 1，将输出 x 的值

D）有语法错误，不能执行

8．下列说法中不正确的是（　　）。

A）程序中定义的变量类型尽量要具有现实意义

B）初学者应有限制地使用多级指针

C）结构体类型的定义最好放在所有函数之前

D）typedef 命令可以用来定义一种新的数据类型

9．下列说法中不正确的是（　　）。

A）定义标识符时要尽量做到见名知意

B）定义较长的标识符时，驼峰命名法是一种较好的命名方法

C）定义变量名时不需要附加变量的类型

D）表示一些常量时可以使用符号常量

10．下列说法中不正确的是（　　）。

A）输入数据之前应有提示信息说明输入数据的个数及类型等

B）输入数据的格式应简洁、统一

C）输出数据时应有信息说明数据的含义

D）输出数据的格式应简洁，不需要加说明信息

二、填空题

1．按照软件工程的思想，系统设计包括 4 个方面的内容，分别是_____、_____、_____、_____。

2．模块设计的基本原则是“_____”。

3．C 语言中用_____实现模块的定义。

4．模块的组装既涉及多个_____的连接问题，也涉及实现具体模块的_____之间的连接调用关系。

5．使用_____预处理命令来实现多个源程序文件之间的连接。

6．文件包含是指_____。

7．若想使用 C 语言提供的数学计算方面的系统功能，应包含的头文件是_____。

8．模块函数间的连接可以分为_____连接和_____连接。

9．长久连接是通过_____或_____和其他模块之间产生的连接关系。

10．模块间的短暂连接以三种形式存在：_____、_____和_____。

11．模块间的_____使得模块的功能比较独立，模块调用和组装非常灵活；而模块间的_____使得模块间衔接紧密，模块间的耦合加强。

12．全局变量属于公有变量，可以被许多函数访问使用，但注意当函数内部有与全局变量同名的局部变量时，系统默认使用的是_____。

13．一般情况下，源程序中的所有内容都参加编译，但有时希望其中一部分内容只在满足一定条件下进行编译，也就是对一部分内容指定编译条件。这种编译方式称为_____。

14．定义变量或函数名时应尽量做到“_____”。

15．当变量名或函数名由一个或多个单词连接在一起，第一个单词以小写字母开始，其后每个单词的首字母大写，这种标识符命名方式称为_____。

16．为了帮助阅读程序，应当在程序中使用_____。

17．书写程序时，为了使程序的层次结构清晰，应当使用_____格式。

三、简答题

1．使用模块化方法开发程序有什么好处？

2．什么是短暂连接？什么是长久连接？

3．简述模块设计的标识符风格。

4．条件编译有哪几种形式？

附　录

附录 A　常用 C 语言库函数

A.1　数学函数

在使用数学函数时，应该在该源文件中使用以下命令行：

#include <math.h>　或　#include "math.h"

表 A.1　math.h 函数列表

函 数 名	函 数 原 型	功　能	返 回 值	说　明
abs	int abs(int x);	求整数 x 的绝对值	计算结果	
acos	double acos(double x);	计算 arccos(x)的值	计算结果	$-1 \leqslant x \leqslant 1$
asin	double asin(double x);	计算 arcsin(x)的值	计算结果	$-1 \leqslant x \leqslant 1$
atan	double atan(double x);	计算 arctan(x)的值	计算结果	
atan2	double atan2(double x,double y);	计算 arctan(x/y)的值	计算结果	
cos	double cos(double x);	计算 cos(x)的值	计算结果	x 单位为弧度
cosh	double cosh(double x);	计算 x 的双曲余弦值	计算结果	
exp	double exp(double x);	计算 e^x 的值	计算结果	
ceil	double ceil(double x);	计算不小于 x 的最小整数	计算结果	
fabs	double fabs(double x);	求 x 的绝对值	计算结果	
floor	double floor(double x);	求不大于 x 的最大整数	该整数的双精度实数	
fmod	double fmod(double x,double y);	求整除 x/y 的余数	返回余数的双精度数	
frexp	double frexp(double val,int *eptr);	把双精度数 val 分解为数字部分（尾数）x 和以 2 为底的指数 n，即 val=$x*2^n$，n 存放在 eptr 指向的变量中	返回数字部分 x，$0.5 \leqslant x \leqslant 1$	
log	double log(double x);	求 $\log_e x$，即 $\ln x$	计算结果	
log10	double log10(double x);	求 $\log_{10} x$，即 $\lg^x$	计算结果	
modf	double modf(double val,int *iptr);	把双精度数 val 分解为整数部分和小数部分，把整数部分存在 iptr 指向的单元	val 的小数部分	
pow	double pow(double x,double y);	计算 x^y 的值	计算结果	
sin	double sin(double x);	计算 sin(x)的值	计算结果	x 单位为弧度
sinh	double sinh(double x);	计算 x 的双曲正弦值	计算结果	
sqrt	double sqrt(double x);	计算 $\sqrt{x}$	计算结果	$x \geqslant 0$
tan	double tan(double x);	计算 tan(x)的值	计算结果	x 单位为弧度
tanh	double tanh(double x);	计算 x 的双曲正切值	计算结果	

A.2　输入/输出函数

在使用输入/输出函数时，应该在该源文件中使用以下命令行：

#include <stdio.h>　或　#include "stdio.h"

表 A.2　stdio.h 函数列表

函 数 名	函 数 原 型	功　能	返　回　值	说　明
cleraerr	void clearer(FILE *fp);	清除 fp 指向的文件的错误标志，同时清除文件结束标志	无	
close	int close(int fd);	关闭文件	关闭成功，返回 0；否则返回–1	非 ANSI 标准
creat	int creat(char *filename,int mode);	以 mode 所指定的方式建立文件，文件名为 filename	成功则返回正数，否则返回–1	非 ANSI 标准
eof	int eof(int fd);	判断是否处于文件结束	遇到文件结束，返回 1；否则返回 0	非 ANSI 标准
fclose	int fclose(FILE *fp);	关闭 fp 所指向的文件，释放文件缓冲区	关闭成功，返回 0；否则返回非 0	
feof	int feof(FILE *fp);	检查文件是否结束	遇文件结束符返回非 0，否则返回 0	
ferror	int ferror(FILE *fp);	测试 fp 所指向的文件是否有错	无错返回 0，有错返回非 0	
fflush	int fflush(FILE *fp);	把 fp 指向的文件的所有数据和控制信息存盘	成功返回 0，否则返回非 0	
fgetc	int fgetc(FILE *fp);	从 fp 所指向的文件中取得下一个字符	返回所得到的字符。出错则返回 EOF	
fgets	char *fgets(char *buf, int n, FILE *fp,);	从 fp 指向的文件读取一个长度为 *n*–1 的字符串，存入起始地址为 buf 的空间	成功，返回地址 buf。若遇文件结束或出错，返回 NULL	
fopen	FILE *fopen(char *filename, char *mode);	以 mode 指定的方式打开名为 filename 的文件	成功，返回一个文件指针（文件信息区的起始地址），否则返回 0	
fprintf	int fprintf(FILE *fp,char *format, args,…);	把 args 的值以 format 指定的格式输出到 fp 所指定的文件中	实际输出的字符数	
fputc	int fputc(char ch,FILE *fp);	将字符 ch 输出到 fp 指定的文件中	成功，则返回该字符；否则返回 EOF	
fputs	int fputs(char *str, FILE * fp);	将 str 指定的字符串输出到 fp 所指定的文件	成功，返回 0；否则返回非 0 值	
fread	int fread(char *pt,unsigned size, unsigned n,FILE *fp);	从 fp 所指定的文件中读取长度为 size 的 *n* 个数据项，存到 pt 指向的内存区	返回所读的数据项个数，如遇文件结束或出错则返回 0	
freopen	FILE *freopen(char *fname, char *mode,FILE *fp);	用 fname 所指定的文件替换 fp 所指定的文件，fname 文件的打开方式由 mode 定义	成功，返回文件指针 fp；否则返回 NULL	
fscanf	int fscanf(FILE *fp, char *format, args,…);	从 fp 指定的文件中按 format 给定的格式将输入数据送到 args 所指向的内存单元（args 是指针）	已输入的数据个数	
fseek	int fseek(FILE *fp,long offset, int base);	将 fp 所指向文件的位置指针移到以 base 所指出的位置为基准、以 offset 为位移量的位置	返回当前位置，否则返回–1	
ftell	long ftell(FILE *fp);	返回 fp 所指向的文件中的读写位置	返回 fp 所指向的文件中的读写位置	
fwrite	int fwrite(char *ptr, unsigned size, unsigned n,FILE *fp);	把 ptr 所指向的 n*size 个字节输出到 fp 所指向的文件中	写到文件中的数据项个数	

续表

函 数 名	函 数 原 型	功　能	返 回 值	说　明
getc	int getc(FILE *fp);	从 fp 所指向的文件读入一个字符	返回所读的字符，若文件结束或出错，返回 EOF	
getchar	int getchar(void);	从标准输入设备读取一个字符	所读字符，若文件结束或出错，则返回–1	
gets	char *gets(char *str);	从标准输入设备读取字符串，并把它们放入由 str 指向的字符数组中	成功返回 str，否则返回 NULL	
getw	int getw(FILE *fp);	从 fp 所指向的文件读取下一个字（整数）	输入的整数。若文件结束或出错，返回–1	非 ANSI 标准函数
kbhit	int kbhit( );	判断是否有键被按下	若有键被按下，返回一个非 0 值，否则返回 0	
lseek	long lseek(int fd,long offset, int base);	根据 base 所确定的位置，按 offset 的偏移量调整 fd 所指定的文件中的读写位置	成功返回该文件中的当前位置，否则返回–1	
open	int open(char *filename,int mode);	以 mode 指出的方式，打开已存在的名为 filename 的文件	成功，返回文件号（正数），否则返回–1	非 ANSI 标准函数
printf	int printf(char *format, args,…);	按 format 指向的字符串规定的格式，将输出表列 args 的值输出到标准输出设备	输出字符的个数。若出错，返回负数	format 可以是一个字符串或字符数组起始地址
putc	int putc(int ch,FILE *fp);	把一个字符 ch 输出到 fp 所指定的文件中	输出的字符 ch。若出错，返回 EOF	
putchar	int putchar(int ch);	把字符 ch 输出到标准输出设备	输出的字符 ch。若出错，返回 EOF	
puts	int puts(char *str);	把 str 指向的字符串输出到标准输出设备，将\0 转换为回车换行	成功返回换行符，失败返回 EOF	
putw	int putw(int i,FILE *fp);	将一个整数 i（即一个字）写到 fp 指向的文件中	返回输出的整数，失败返回 EOF	非 ANSI 标准函数
read	int read(int fd,char *buf, unsigned count);	从文件号 fd 所指示的文件中读 count 个字节到由 buf 指示的缓冲区	返回读入的字节个数。如遇文件结束返回 0，出错返回–1	非 ANSI 标准函数
remove	int remove(char *filename);	删除以 filename 为文件名的文件	成功返回 0，失败返回–1	
rename	int rename(char *oldname, char *newname);	把 oldname 所指的文件名改为由 newname 所指的文件名	成功返回 0，失败返回–1	
rewind	void rewind(FILE *fp);	将 fp 指示的文件中位置指针置于文件开头位置，并清除文件结束标志和错误标志	无	
scanf	int scanf(char *format, args,…);	从标准输入设备按 format 指定的格式字符串规定的格式，输入数据给 args 所指向的单元（args 为指针）	读入并赋给 args 的数据个数。遇文件结束返回 EOF，出错返回 0	
tell	long int tell(int fd);	确定文件描述号 fd 所对应的文件位置指示器的当前值	返回文件位置指示器的当前值，若出错返回–1	
setbuf	void setbuf(FILE *fp, char *buf);	说明 fp 将要使用的缓冲区（长度为 buf）。若 buf 置为 NULL，则关闭缓冲区	无	

续表

函 数 名	函 数 原 型	功　能	返 回 值	说　明
setvbuf	int setvbuf(FILE *fp, char *buf, int mode,int size);	为 fp 所指向的文件提供输入输出操作的缓冲区 buf，缓冲区的大小是 size，使用方式为 mode	成功返回 0，失败返回非 0	
sprintf	int sprintf(char *buf, char *format, args,…);	把按 fornat 规定的格式的 args 数据，送到 buf 所指向的数组中	返回实际放进数组中的字符数	
sscanf	int sscanf(char *buf, char *format, args,…);	按 format 规定的格式从 buf 指向的数组中读入数据给 args 所指向的单元（args 为指针）	返回值为实际赋值的个数；若返回 0，则无任何字段被赋值	
tmpnam	char *tmpnam(char *name);	生成一个与目录中其他文件名不同的临时文件名，并把它放入由 name 指向的数组中	成功返回指向 name 的指针，否则返回 NULL 指针	
tmpfile	FILE *tmpfile( );	打开一个临时文件并返回指向这个文件的指针。由该函数产生的临时文件在被关闭或程序结束时会自动被删除	成功返回指向文件的指针，失败返回 NULL	
write	int write(int fd, char *buf, unsigned int size);	把 buf 指向的缓冲区中 size 个字节写到 fd 文件中	返回实际写出的字节数；出错，返回–1	非 ANSI 标准函数

A.3　字符函数

ANSI C 标准要求在使用字符函数时要包含头文件 ctype.h。有的 C 语言编译系统不遵循 ANSI C 标准的规定，而用其他名称的头文件。请使用时查阅相关手册。

表 A.3　ctype.h 函数列表

函 数 名	函 数 原 型	功　能	返 回 值
isalnum	int isalnum(int ch);	检查 ch 是否为字母（alpha）或数字（number）	是字母或数字返回非 0 值，否则返回 0
isalpha	int isalpha(int ch);	检查 ch 是否为字母	是，返回非 0 值；不是则返回 0
iscntrl	int iscntrl(int ch);	检查 ch 是否为控制字符（其 ASCII 码在 0 和 0x1F 之间）	是，返回非 0 值；不是则返回 0
isdigit	int isdigit(int ch);	检查 ch 是否为数字（0～9）	是，返回非 0 值；不是则返回 0
isgraph	int isgraph(int ch);	检查 ch 是否为可打印字符（其 ASCII 码在 0x21 到 0x7E 之间），不包括空格	是，返回非 0 值；不是则返回 0
islower	int islower(int ch);	检查 ch 是否为为小写字母（a～z）	是，返回非 0 值；不是则返回 0
isprint	int isprint(int ch);	检查 ch 是否为可打印字符（其 ASCII 码在 0x20 到 0x7E 之间），包括空格	是，返回 1；否则返回 0
ispunct	int ispunct (int ch);	检查 ch 是否为标点符号（不包括空格），即除字母、数字和空格以外的所有可打印字符	是，返回 1；否则返回 0
isspace	int isspace(int ch);	检查 ch 是否为空格、制表符或换行符	是，返回 1；否则返回 0
isupper	int isupper(int ch);	检查 ch 是否为大写字母（A～Z）	是，返回非 0 值；不是则返回 0
isxdigit	int isxdigit(int ch);	检查 ch 是否为一个 16 进制数学字符（即 0～9，或 A～F，或 a～f）	是，返回非 0 值；不是则返回 0
tolower	int tolower(int ch);	将 ch 字符转换为小写字母	返回 ch 所代表的字符的小写字母
toupper	int toupper(int ch);	将 ch 字符转换为大写字母	返回 ch 所代表的字符的大写字母

A.4　字符串函数

在使用字符串函数时，应包含头文件 string.h。

表 A.4　string.h 函数列表

函 数 名	函 数 原 型	功　能	返 回 值
memchr	void memchr(void *buf,int ch, unsigned int count);	在 buf 的前 n 个字符里搜索 ch 第一次出现的位置	返回指向 buf 中 ch 第一次出现的位置的指针，未找到则返回 NULL
memcmp	int memcmp(void *buf1, void *buf2, unsigned int count);	按字典顺序比较由 buf1 和 buf2 指向的数组的前 count 个字符	buf1 小于 buf2 时，返回一个负整数；buf1 等于 buf2 时，返回 0；buf1 大于 buf2 时，返回一个正整数
memcpy	void *memcpy(void *to, void *from, unsigned int count);	把 from 指向的数组中的前 count 个字符复制到 to 指向的数组中	返回指向 to 的指针
memmove	void *memmove(void *to, void *from,unsigned int count);	从 from 指向的数组中把前 count 个字符复制到 to 指向的数组中	返回指向 to 的指针
memset	void *memset(void *buf,int ch, unsigned int count);	将 buf 中的前 count 个字符设置为 ch	返回 buf
strcat	char *strcat(char *str1,char *str2);	把字符串 str2 连接到 str1 后面，原 str1 后面的\0 被删除	返回 str1
strchr	char *strchr(char *str, int ch);	找出 str 指向的字符串中第一次出现字符 ch 的位置	返回指向该位置的指针，未找到则返回 NULL
strcmp	int strcmp(char *str1,char *str2);	比较两个字符串 str1 和 str2。从第一个字符开始比较，直到遇到不同的字符或已到串尾	str1>str2，返回正整数；str1=str2，返回 0；str1<str2，返回负整数
strcpy	char *strcpy(char *str1,char *str2);	把 str2 指向的字符串复制到 str1 中	返回 str1
strlen	unsigned int strlen(char *str);	统计字符串 str 中字符的个数（不包括终止符'\0'）	返回字符个数
strcspn	int strcspn(char *str1,char *str2);	确定 str1 中出现的属于 str2 的第一个字符的下标	返回字符下标
strncat	char *strncat(char *str1, char *str2, unsigned int count);	把 str2 指向的字符串中最多 count 个字符连到 str1 后面，并用 NULL 结尾	返回 str1
strncmp	int strncmp(char *str1,char *str2, unsigned int count);	按字典顺序比较两个以 NULL 结尾的字符串中最多 count 个字符	str1>str2，返回正整数；str1=str2，返回 0；str1<str2，返回负整数
strncpy	char *strncpy(char *str1, char *str2, unsigned int count);	把 str2 中最多 count 个字符复制到 str1 中去	返回 str1
strpbrk	char *strcspn(char *str1, char *str2);	确定 str1 中第一个与 str2 中任何一个字符相匹配的指针位置	返回 str1 中第一个与 str2 中任何一个字符相匹配的字符指针。如果未找到则返回空
strspn	int strspn(char *str1, char *str2);	确定 str1 中出现的属于 str2 的第一个字符的下标	返回 str1 中属于 str2 的第一个字符的下标
strstr	char *strstr(char *str1, char *str2);	查找 str2 指向的字符串在 str1 指向的字符串中第一次出现的位置	子串首次出现的地址。如果找不到，则返回空指针 NULL

A.5　动态存储分配函数

ANSI C 标准建议设 4 个有关的动态存储分配的函数，即 calloc、malloc、free、realloc。实际上，许多

C 语言编译系统在实现时往往增加了一些其他函数。ANSI 建议在头文件 stdlib.h 中包含有关信息，但许多 C 语言编译要求用 malloc.h。因此，在使用时应查阅相关手册。

ANSI 标准要求动态分配系统返回 void 指针。void 指针具有一般性，它们可以指向任何类型的数据。但目前有的 C 语言编译系统所提供的这类函数返回 char 指针。无论以上哪种情况，都需要用强制类型转换的方法把 void 或 char 指针转换成所需的类型。

表 A.5　mallo.h 函数列表

函　数　名	函 数 原 型	功　　能	返　回　值
calloc	void *calloc(unsigned int n, unsigned int size);	分配 *n* 个数据项的连续内存空间，每个数据项的大小为 size	分配内存单元的起始地址。如分配不成功，则返回 NULL
free	void free(void *p);	释放 p 所指向的内存区	无
malloc	void *malloc(unsigned int size);	分配 size 个字节的存储区	返回所分配内存区的起始地址。若内存不够，返回 NULL
realloc	void *realloc(void *p, unsigned int size);	将 p 所指出的已分配内存区的大小改为 size。size 可以比原来分配的空间大或小	返回指向该内存区的指针

A.6　时间函数

当需要使用系统日期和时间函数时，需要头文件 time.h。其中定义了 3 个类型：clock_t 和 time_t 用来表示系统的时间和日期。结构体类型 tm 把日期和时间分解成为它的成员。tm 结构体类型的定义如下：

```
struct tm{
    int    tm_sec;          /* 秒，0～59 */
    int    tm_min;          /* 分，0～59 */
    int    tm_hour;         /* 小时，0～23 */
    int    tm_mday;         /* 天，0～31 */
    int    tm_mon;          /* 从一月开始的月数，0～11 */
    int    tm_year;         /* 自 1990 开始的年数 */
    int    tm_wday;         /* 自星期日开始的天数，0～6 */
    int    tm_yday;         /* 自 1 月 1 日起的天数，0～365 */
    int    tm_isdst;        /* 夏季时间标志 */
};
```

另外，用于存放时间和日期的结构体的定义为：

```
struct time{
    unsigned char ti_min;       /* 分 */
    unsigned char ti_hour;      /* 时 */
    unsigned char ti_hund;      /* 百分之一秒 */
    unsigned char ti_sec;       /* 秒 */
};
struct date{
    int da_year;                /* 年 */
    char da_day;                /* 日 */
    char da_mon;                /* 月，1～12 */
};
```

表 A.6　time.h 函数列表

函　数　名	函 数 原 型	功　　能	返 回 值
asctime	char *asctime(struct tm *p);	将日期和时间转换成 ASCII 字符串	返回一个指向字符串的指针
clock	clock_t clock( );	确定程序运行到现在所花费的时间（用返回值除以宏 CLK_TCK 得到的时间为秒数）	返回程序开始到该函数被调用所花费的时间。若失败，则返回–1

续表

函 数 名	函 数 原 型	功　能	返 回 值
ctime	char *ctime(time_t *time);	把日期和时间转换成字符串	返回指向包含日期和时间的字符串指针
difftime	double difftime(time_t time2, time_t time1);	计算 time1 和 time2 之间所差的秒数	返回两个时间的双精度差值
getdate	void getdate(struct date *datep);	取得系统的当前日期，放在 datep 所指向的结构体中（需要包含的头文件是 dos.h）	无
gettime	void gettime(struct time *timep);	取得系统的当前时间，放在 datep 所指向的结构体中（需要包含的头文件是 dos.h）	无
gmtime	struct tm *gmtime(time_t *time);	将日期和时间转换为格林尼治标准时间	返回指向结构体 tm 的指针
setdate	void setdate(struct date *datep);	将系统日期设置为 datep 所指向的结构体中定义的日期	无
settime	void settime(struct time *timep);	将系统时间设置为 timep 所指向的结构体中定义的时间	无
time	time_t time(time_t time);	取得系统的当前时间	返回系统的当前日历时间。若无系统时间，返回–1

A.7　其他函数

这里列出的函数都是标准库函数，但却不容易归到前面的某一类中。使用这些函数要包含头文件 stdlib.h。这个文件定义了两个类型 div_t 和 ldiv_t，定义如下：

```
typedef struct{
        int   quot;
        int   rem;
}div_t;
typedef struct{
        long   quot;
        long   rem;
}ldiv_t;
```

表 A.7　stdlib.h 函数列表

函 数 名	函 数 原 型	功　能	返 回 值
abort	void abort( );	立刻结束程序运行，不清理任何文件缓冲区	无
atof	double atof(char *str);	把 str 指向的字符串转换成一个 double 值	返回双精度计算结果
atoi	int atoi(char *str);	将 ASCII 字符串转换为整数	返回转换结果
atol	long atol(char *str);	将 str 指向的 ASCII 值符串转换为长整型值	返回转换结果。若不能转换，返回 0
bsearch	void *bsearch(void *key, void *base,unsigned int num, unsigned int size,int (*compare)( ));	对一个 base 指向的已排好序的数组进行二分查找。数组的元素个数是 num，每一元素的大小为 size 字节。compare 指向的函数用来把数组元素与关键字进行比较	返回一个指向匹配 key 所指向的关键字的第一个成员的指针。若没有找到，则返回 NULL
exit	void exit(int status);	使程序立刻正常终止。status 的值传给调用过程	无

续表

函 数 名	函 数 原 型	功　能	返 回 值
div	div_t div(int num, int denom);	计算 num/denom	返回计算的商和余数
itoa	char *itoa(int num,char *str, int radix);	把整数 num 转换成与其等价的字符串，并把结果放在 str 指向的字符串中，由 radix 决定在转换成输出串时所采用的进制数	返回一个指向 str 的指针
ldiv	ldiv_t ldiv(long int num, long int denom);	计算 num/denom	返回商和余数
ltoa	char *ltoa(long num,char *str,int radix);	把长整数 num 转换成与其等价的字符串，并把结果放在 str 指向的字符串中，由 radix 决定在转换成输出串时所采用的进制数	返回一个指向 str 的指针
qsort	void qsort(void *base, unsigned int num, unsigned int size,int (*comp)( ));	反复调用 comp 所指向的由用户自己编写的比较函数，对 base 指向的数组进行排序。num 是数组的元素个数，size 是数组元素的字节数	无
rand	int rand( );	产生随机数	返回 0 到 RAND_MAX 之间的整数。RAND_MAX 是返回的最大可能值，在头文件中定义
strtod	double strtod(char *start, char **end);	把存储在 start 指向的数字字符串转换成 double 类型，直到出现不能转换成浮点数的字符为止，剩余的字符串赋给指针 end	返回转换结果。若未进行转换，则返回 0。若转换发生错误，则返回 HUGE_VAL 或_HUGE _VAL 表示上溢或下溢
strtol	long int strtol(char *start, char **end,int radix);	把 start 指向的数字字符串转换成 long int 类型，直到出现不能转换成长整数的字符为止，剩余的字符串赋给指针 end，数字的进制由 radix 确定	返回转换结果。若未进行转换，则返回 0。若转换发生错误，则返回 LONG_MAX 或 LONG_MIN 表示上溢或下溢
strtoul	unsigned long int strtoul (char *start, char **end,int radix);	把 start 指向的数字字符串转换成 unsigned long int 类型，直到出现不能转换成 unsigned long int 的字符为止，剩余的字符串赋给指针 end，数字的进制由 radix 确定	返回转换结果。若未进行转换，则返回 0。若转换发生错误，则返回 ULONG_MAX 或 ULONG _MIN 表示上溢或下溢
system	int system(char *str);	把 str 指向的字符串作为一个命令传送到操作系统的命令处理程序中	返回值依赖于不同的编译版本。通常，成功执行返回 0，否则返回一个非 0 值

附录 B　ASCII 码表

信息在计算机上是用二进制表示的。因此，计算机上都配有输入和输出设备，这些设备的主要目的就是，以一种人类可阅读的形式将信息在这些设备上显示出来供人阅读理解。为保证人类和设备，设备和计算机之间能进行正确的信息交换，人们编制了统一的信息交换代码，这就是 ASCII 码表，它的全称是“美国信息交换标准代码”。

表 B.1　ASCII 码表

八 进 制	十六进制	十 进 制	字　符	八 进 制	十六进制	十 进 制	字　符
00	00	0	NUL	36	1e	30	RS
01	01	1	SOH	37	1f	31	US

续表

八　进　制	十六进制	十　进　制	字　　符	八　进　制	十六进制	十　进　制	字　　符
02	02	2	STX	40	20	32	SP
03	03	3	ETX	41	21	33	!
04	04	4	EOT	42	22	34	"
05	05	5	ENQ	43	23	35	#
06	06	6	ACK	44	24	36	$
07	07	7	BEL	45	25	37	%
10	08	8	BS	46	26	38	&
11	09	9	HT	47	27	39	`
12	0a	10	LF	50	28	40	(
13	0b	11	VT	51	29	41	)
14	0c	12	FF	52	2a	42	*
15	0d	13	CR	53	2b	43	+
16	0e	14	SO	54	2c	44	,
17	0f	15	SI	55	2d	45	–
20	10	16	DLE	56	2e	46	.
21	11	17	DC1	57	2f	47	/
22	12	18	DC2	60	30	48	0
23	13	19	DC3	61	31	49	1
24	14	20	DC4	62	32	50	2
25	15	21	NAK	63	33	51	3
26	16	22	SYN	64	34	52	4
27	17	23	ETB	65	35	53	5
30	18	24	CAN	66	36	54	6
31	19	25	EM	67	37	55	7
32	1a	26	SUB	70	38	56	8
33	1b	27	ESC	71	39	57	9
34	1c	28	FS	72	3a	58	:
35	1d	29	GS	73	3b	59	;
74	3c	60	<	136	5e	94	^
75	3d	61	=	137	5f	95	_
76	3e	62	>	140	60	96	'
77	3f	63	?	141	61	97	a
100	40	64	@	142	62	98	b
101	41	65	A	143	63	99	c
102	42	66	B	144	64	100	d
103	43	67	C	145	65	101	e
104	44	68	D	146	66	102	f
105	45	69	E	147	67	103	g
106	46	70	F	150	68	104	h
107	47	71	G	151	69	105	i
110	48	72	H	152	6a	106	j
111	49	73	I	153	6b	107	k
112	4a	74	J	154	6c	108	l
113	4b	75	K	155	6d	109	m
114	4c	76	L	156	6e	110	n
115	4d	77	M	157	6f	111	o

续表

八　进　制	十六进制	十　进　制	字　　符	八　进　制	十六进制	十　进　制	字　　符
116	4e	78	N	160	70	112	p
117	4f	79	O	161	71	113	q
120	50	80	P	162	72	114	r
121	51	81	Q	163	73	115	s
122	52	82	R	164	74	116	t
123	53	83	S	165	75	117	u
124	54	84	T	166	76	118	v
125	55	85	U	167	77	119	w
126	56	86	V	170	78	120	x
127	57	87	W	171	79	121	y
130	58	88	X	172	7a	122	z
131	59	89	Y	173	7b	123	{
132	5a	90	Z	174	7c	124	|
133	5b	91	[	175	7d	125	}
134	5c	92	\	176	7e	126	~
135	5d	93	]	177	7f	127	del

表 B.2　0～32 对应字符的含义

	缩　　写	字　　符	含　　义		缩　　写	字　　符	含　　义
0	NUL	null	空字符	17	DC1	device control 1	设备控制 1
1	SOH	start of headling	标题开始	18	DC2	device control 2	设备控制 2
2	STX	start of text	正文开始	19	DC3	device control 3	设备控制 3
3	ETX	end of text	正文结束	20	DC4	device control 4	设备控制 4
4	EOT	end of transmission	传输结束	21	NAK	negative acknowledge	拒绝接收
5	ENQ	enquiry	请求	22	SYN	synchronous idle	同步空闲
6	ACK	Acknowledge	收到通知	23	ETB	end of trans. block	传输块结束
7	BEL	Bell	响铃	24	CAN	cancel	取消
8	BS	backspace	退格	25	EM	end of medium	介质中断
9	HT	horizontal tab	水平制表符	26	SUB	substitute	替补
10	LF	line feed, new line	换行键	27	ESC	escape	溢出
11	VT	vertical tab	垂直制表符	28	FS	file separator	文件分割符
12	FF	form feed, new page	换页键	29	GS	group separator	分组符
13	CR	carriage return	回车键	30	RS	record separator	记录分离符
14	SO	shift out	不用切换	31	US	unit separator	单元分隔符
15	SI	shift in	启用切换	32	SP	space	空格
16	DLE	data link escape	数据链路转义				

附录 C　C 语言运算符的优先级与结合性

优　先　级	运　算　符	含　　义	参与运算对象的数目	结　合　性
1	( ) [ ] –> .	圆括号运算符 下标运算符 指向结构体成员运算符 结构体成员运算符		自左至右

续表

优 先 级	运 算 符	含　义	参与运算对象的数目	结 合 性
2	! ~ ++ — － (类型) * & sizeof	逻辑非运算符 按位取反运算符 自增运算符 自减运算符 负号运算符 类型转换运算符 指针运算符 取地址运算符 求类型长度运算符	单目运算符	自右至左
3	* / %	乘法运算符 除法运算符 求余运算符	双目运算符	自左至右
4	＋ －	加法运算符 减法运算符	双目运算符	自左至右
5	<< >>	左移运算符 右移运算符	双目运算符	自左至右
6	< <= > >=	关系运算符	双目运算符	自左至右
7	＝＝ ！＝	等于运算符 不等于运算符	双目运算符	自左至右
8	&	按位与运算符	双目运算符	自左至右
9	^	按位异或运算符	双目运算符	自左至右
10	|	按位或运算符	双目运算符	自左至右
11	&&	逻辑与运算符	双目运算符	自左至右
12	||	逻辑或运算符	双目运算符	自左至右
13	？：	条件运算符	三目运算符	自右至左
14	＝ ＋＝ －＝ *＝ /＝ %＝ >>= <<= &= ^= |=	赋值运算符	双目运算符	自右至左
15	,	逗号运算符 （顺序求值运算符）		自左至右

按运算符优先级从高到低为：单目运算符→双目运算符→三目运算符→赋值运算符→逗号运算符。特别是在双目运算符中，按运算符优先级从高到低为：算术运算符→移位运算符→关系运算符（其中“==”和“!=”优先级又较低）→逻辑运算符（按位与→按位异或→按位或→逻辑与→逻辑或）。

参 考 文 献

[1] 谭浩强．C 程序设计（第二版）．北京：清华大学出版社，1999

[2] 谭浩强．C 程序设计题解与上机指导（第二版）．北京：清华大学出版社，2000

[3] 顾治华，陈天煌，忽朝俭．C 语言程序设计．北京：机械工业出版社，2008

[4] James W.McCord．Borland C++ 3.1 程序员参考手册．张素琴，李景淑，李旭．北京：清华大学出版社，1995

[5] Herbert Schildt．C 语言大全（第二版）．戴健鹏．北京：电子工业出版社，1995

[6] 网冠科技．C 语言时尚编程百例．北京：机械工业出版社，2004

[7] 谭浩强，张基温，唐永炎．C 语言程序设计教程（第二版）．北京：高等教育出版社，1998

[8] 李崇泰．C 语言案例教程．北京：电子工业出版社，2005

[9] 杨旭，王爱赜．C 语言程序设计实用教程．北京：人民邮电出版社，2005

[10] Delores M. Etter．工程问题 C 语言求解．朱剑平，付宇光．北京：清华大学出版社，2005